Volume I

Ada Implementation Guide

March 1992

DEPARTMENT OF THE NAVY

Naval Information Systems Management Center
Space and Naval Warfare Systems Command

REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1992	Final

4. TITLE AND SUBTITLE
Ada Implementation Plan: Department of the Navy. Volumes I [redacted]

5. FUNDING NUMBERS
N/A

6. AUTHOR(S)
Naval Information Systems Management Center
Space and Naval Warfare Systems Command

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Information Systems Management Center
Space and Naval Warfare Systems Command

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Department of Navy
Naval Information Systems Management Center
Washington, DC 20360-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unlimited Distribution

12b. DISTRIBUTION CODE
Unlimited

13. ABSTRACT *(Maximum 200 words)*

This two-volume document discusses the use of Ada for software system development in the DoN. It contents apply to both AIS and MCCR communities. The DoN prepared this report ot provide guidance to Navy Program Managers and their staffs as they implement the (1) DoD Appro priations Act, 1992 Public Law 02-172 (Nov 26, 1991), 105 Stat. 1188-1189 and (2) ASN (RDA) memo, Interim Department of the Navy Policy on Ada of 24 June 91 (NOTAL).

14. SUBJECT TERMS
Ada programming language, Navy, systems development, software development, software engineering, Ada9X, policy

15. NUMBER OF PAGES
124

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unclassified

Volume I

Ada Implementation Guide

March 1992

DEPARTMENT OF THE NAVY

Naval Information Systems Management Center
Space and Naval Warfare Systems Command

Acknowledgements

The following DON personnel contributed their valuable time, insight, and perspective in developing this guide. Their contributions and hard work were invaluable and are greatly appreciated.

Chair:	A.D. Stuart, NISMC
Deputy Chair:	CDR Martin Romeo, SPAWAR 224-1

Mr. Currie Colket	AJPO
Mr. George Robertson	NCCOSC RDTE DIV, San Diego
Ms. Shirley Peele	NAVSURFWARCEN FLTCOMBATDIRSSACT, Dam Neck
Mr. Guy Taylor	NAVSURFWARCEN FLTCOMBATDIRSSACT, Dam Neck
Mr. Lester Hummel	FMSO
Mr. Hank Stuebing	NAVAIRWARCENACDIV, Warminster
Mr. Chuck Koch	NAVAIRWARCENACDIV, Warminster
Mr. John McLaurin	NADEP
Mr. Tom Coyle	NAVAIR
Mr. Jim Welch	NAVCOMTSSA
Mr. Greg Engledove	NAVSEA
Ms. Joan McGarity	NCTC
Mr. Robert Calland	NCCOSC RDTE DIV, San Diego
Ms. Cathy Ruiz	NCCOSC RDTE DIV, San Diego
Mr. Rich Bergman	NCCOSC RDTE DIV, San Diego
Ms. Donna K. Fisher	NCCOSC RDTE DIV, San Diego
Mr. Dan Green	NAVSURFWARCENDIV, Dahlgren
Mr. Frank Ervin	NAVSURFWARCENDIV, Dahlgren
Mr. Eugene Hodgson	NAVSURFWARCENDIV, Dahlgren
Mr. Charles Flemmings	NAVSURFWARCENDIV, Dahlgren
Mr. Tom Conrad	NAVUNSEAWARCENDIV, Newport
Mr. Ron House	NAVUNSEAWARCENDIV, Newport
Major J. Spegele	USNA
Major Dave Thompson	USMC
Capt Gerald DePasquale	USMC
Mr. Doug Afdahl	USNA
Mr. Jim Moss	USNA
Ms. Patricia Grandy	NARDAC, San Francisco
Mr. Bond Wetherbe	NAVCOMTELSTA
Mr. George Frilot	NAVCOMTELSTA

LCDR Jean Shkapsky	Naval Postgraduate School
Ms. Kathy Steele	NAVAIRWARCENACDIV, Patuxent River
Mr. John Shields	NAVAIRWARCENACDIV, Patuxent River
LCDR Anne Sullivan	BUPERS

We also wish to express our thanks to the document coordinator, Ms. Susan Scott, with support from Ms. Salvi Mugol, and the editor, Ms. Madeline Nevins, all of Booz·Allen & Hamilton Inc.

Contents

VOLUME I

Section 1: INTRODUCTION 1
1.1 Background 1
1.2 Scope of Ada Use 1
1.3 Experience of Ada Use 2
1.4 Document Scope 4
1.5 Content and Organization of This Document 5

Section 2: Ada POLICY 7
2.1 Policy Directives 7
2.2 Policy Rationale 7
2.3 Policy Description 7

Section 3: IMPLEMENTATION GUIDANCE 15
3.1 Program Planning 15
3.1.1 Organizational Structure 16
3.1.2 Resources 16
3.1.3 Personnel and Training 17
3.1.4 Technology and Tools 17
3.1.5 Program-Specific Standards and Procedures 18
3.1.6 Software Development Process 19
3.1.7 Life-Cycle Maintainability 24
3.2 Acquisition Planning 24
3.2.1 Acquisition Plan 25
3.2.2 Statement of Work 25
3.2.3 Proposal Preparation Instructions 26
3.2.4 Proposal Evaluation Criteria 26
3.2.5 Government Estimate 27
3.2.6 Deliverables—Contract Data Requirements List 28
3.3 System Engineering and Risk Management 28
3.3.1 Software Versus Hardware in a System Context 29
3.3.2 Prototyping 29
3.3.3 Project Context Benchmarks 29
3.3.4 Requirements Volatility and Traceability 30
3.3.5 Support Software Acquisition Impacts 30
3.3.6 Ada Software Reuse 30
3.3.7 Prime Contractor-Subcontractor Relationships 31

3.3.8 Testing Philosophy and Methodology 31
3.3.9 Integration Philosophy 32
3.3.10 Incremental Development 32
3.4 Software Engineering 33
3.4.1 Ada Software Engineering Goals 33
3.4.2 Software Engineering Principles 34
3.4.3 Ada Adaptation Guidelines 35
3.5 Test and Evaluation 38
3.6 Post-Deployment Maintenance 38
3.7 Highlights 39

Section 4: ENVIRONMENTS 43
4.1 Software Engineering Environment 43
4.2 Typical Tool Set 44
4.3 Commercial Ada Development Tools 45
4.4 Mission-Critical Computer Resources Environment 46
4.4.1 Standard Embedded Computer Resources 46
4.4.2 Commercial Computer Resources 46
4.4.3 Next Generation Computer Resources 47
4.5 Automated Information Systems Environment 47
4.6 Project Support Environments 47
4.6.1 Commercial Ada Environments 47
4.6.2 AdaSAGE 48
4.6.3 Ada Language System/Navy 50
4.6.4 Related PSE Activities and Issues 50
4.7 Post-Deployment Software Support 54

Section 5: Ada TECHNOLOGY ISSUES 57
5.1 Ada Transition 57
5.1.1 Ada Upgrade Opportunities 57
5.1.2 Mixing Ada With Other Languages 59
5.1.3 Reengineering 59
5.1.4 Reverse Engineering 60
5.1.5 Porting or Portability 60
5.2 Compiler Selection 61
5.2.1 Validation 62
5.2.2 Evaluation 62
5.2.3 Benchmarks 63
5.3 Interface Standards 64
5.3.1 Bindings 65
5.3.2 Secondary Standards 66
5.3.3 Important Standardization Areas 66

5.4 Software Reuse 69
5.4.1 Economic Benefits of Software Reuse 69
5.4.2 Software Reuse Techniques Applicable to Ada 70
5.4.3 Management Issues 71
5.4.4 Reuse Repositories 72
5.5 Life-Cycle Documentation 72
5.5.1 DOD-STD-2167A 72
5.5.2 DOD-STD-7935A 73

Section 6: LESSONS LEARNED 75
6.1 Standards and Policy 76
6.2 Project Management 76
6.3 Development Process 76
6.4 Corporate Knowledge and Software Development Experience 77
6.5 Training 77
6.6 Resources and Facilities 77
6.7 Tools 77
6.8 Reuse 77
6.9 Project Costs 78

Section 7: FUTURE DIRECTIONS 79
7.1 Ada Technology Insertion Program 80
7.2 Ada 9X 81
7.2.1 Background 81
7.2.2 Ada 9X Transition Activities 82
7.3 Ada Reuse 84
7.4 Ada Language System/Navy 85
7.5 Next Generation Computer Resources 85
7.5.1 Project Support Environment Standard Working Group 86
7.5.2 POSIX Standard Working Group 87
7.6 Common Ada Programming Support Environment Interface Set 87
7.7 Portable Common Interface Set 88
7.8 Software Technology for Adaptable, Reliable Systems 89
7.8.1 Reuse 89
7.8.2 Process 90
7.8.3 Environment 90
7.8.4 Demonstration 90
7.9 Corporate Information Management 90
7.10 Software Engineering Institute 91
7.10.1 Technology Division 92
7.10.2 Products and Services Division 96

7.11 Plans 98
7.11.1 Software Action Plan 98
7.11.2 Software Technology Plan 99
7.11.3 Computer Resources Strategy 99

GLOSSARY 101

REFERENCES 111

VOLUME II

Appendix A: Helpful Sources A-1
A.1 Government Sources A-1
A.1.1 Organizations A-2
A.1.2 Training A-7
A.1.3 Publications A-10
A.1.4 Bulletin Boards A-15
A.1.5 Repositories A-17
A.1.6 Conferences and Special Interest Groups A-23
A.1.7 Operational Support Development Tools A-24
A.2 Ada Information Clearinghouse A-25
A.2.1 Public Access to the AdaIC Bulletin Board A-27
A.2.2 Access to Ada Information on the Defense Data Network .. A-28
A.2.3 Info_Ada Digest A-29
A.2.4 Document Reference Sources A-30
A.2.5 AdaIC File Directory A-30
A.3 Other Sources A-40
A.3.1 Training A-40
A.3.2 Publications A-41
A.3.3 Repositories A-44
A.3.4 Conferences and Special Interest Groups A-46
A.3.5 Operational Support Development Tools A-47

Appendix B: Supplementary Reading B-1
Appendix C: Initial Process C-1
Appendix D: Source Line of Code Metrics Definition D-1
Appendix E: Example of Metric Wording for Use in a Contractual Document E-1
Appendix F: Software Tool Descriptions F-1
Appendix G: Ada Bindings and Secondary Standards G-1
G.1 Portable Operating System Interface for UNIX G-1
G.2 Structured Query Language G-3
G.3 XWindows G-6
G.4 Government Open System Interconnection Profile G-7
G.5 Graphics Standards G-9

Appendix H: Ada Binding Products H-1
Appendix I: Lessons Learned I-1
I.1 Advanced Field Artillery Tactical Data System I-13
I.2 AN/BSY-2 ..... I-14
I.3 Ada Language System/Navy I-19
I.4 Avionics Project I-21

I.5 PEO-SSAS, PMS-414, SEA LANCE I-23
I.6 Navy World Wide Military Command and Control System (WWMCCS) Site-Unique Software (NWSUS) Project Mission I-26
I.7 Event-Driven Language/COBOL-to-Ada Conversion Program I-29
I.8 Shipboard Gridlock System With Auto-Correlation I-30
I.9 Combat Control System MK2 ..... I-32
I.10 P-3C Update IV Ada Development I-34
I.11 Standard Financial System Redesign I-38
I.12 Reconfigurable Mission Computer Project I-41
I.13 Intelligent Missile Project I-43

Appendix J: FY91 Ada Technology Insertion Program Projects J-1
J.1 Education J-1
J.2 Bindings J-1
J.3 Technology J-4

Appendix K: Navy and Marine Corps Ada Projects K-1
Appendix L: Glossary L-1

List of Tables and Figures

2-1 DON Directives and Instructions for Implementing Public Law 102-172 . 8
2-2 DON Ada Policy Implementation Matrix 9
7-1 ACVC Schedule 84

Section 1
Introduction

Public Law 102-172, Section 8073 of the Department of Defense (DOD) Appropriations Act, 1992, enacted on November 26, 1991, requires that, "where cost effective, all Department of Defense software shall be written in the programming language Ada. . ." The Department of the Navy (DON) has developed policies and standards for using Ada and prepared this guide to help Program Managers and their staffs to implement these policies. (For the purpose of this document, Program Managers and Project Managers are synonymous.)

1.1 BACKGROUND

Ada was developed to control the proliferation of programming languages, establish a standard programming language for DOD, and reduce DOD's cost to maintain mission-critical computer software. The evolution of Ada as DOD's standard programming language began with the Ironman requirements document. The evolution continued with the selection of the Green Language, the adoption of Military Standard (MIL-STD)-1815A in 1983, the DeLauer Memo, and DOD Directives 3045.1 and 3405.2; it culminated in the passage of Public Law 102-172.

1.2 SCOPE OF Ada USE

For most DOD software application development, Ada will provide benefits in cost, schedule, and quality across the software life cycle. In addition, the use of a single High Order Language (HOL) within the DOD provides superior supportability of DOD requirements. Consequently, Congress has recognized that Ada is the language of choice for DOD software application development and has mandated its use unless an alternative approach can be demonstrated to be cost-effective over the application life cycle. Ada must be used during the concept exploration and definition, demonstration and validation, engineering and manufacturing, development, and production/deployment phases of a system and for both application and support software. In addition, Ada must be used for any major modifications to existing systems and for any systems that involve integrating components into systems that incorporate commercial and other nondevelopmental software. To use an alternative programming language, a waiver must first be obtained from the DON Software Executive Official; the waiver must document that the alternative approach will be significantly more cost-effective over the life cycle. (For procedures for obtaining a waiver, please refer to the DON Ada Policy.) Both the Mission-Critical Computer Resources (MCCR) and Automated Information Systems (AIS) communities are subject to these requirements.

The MCCR community consists of organizations working on computer resources critical to the conduct of the military mission of the DON, including those for all tactical and strategic weapons, communications, command and control, cryptologic activities related to national security, and intelligence systems that directly support military operations. Such systems often are embedded. The MCCR community has been developing software systems in Ada since 1985; however, most systems development has been contracted to the private sector.

The AIS community comprises organizations working on business computer resources that are not mission critical, including all administrative, logistics, financial, personnel, and workload planning systems. Such systems may operate on microcomputers or mainframe computers in a stand-alone or networked mode. The AIS community has been developing software systems in a variety of HOLs. Relatively few AIS application systems have been developed in Ada, and few field activities or claimants have Ada training plans in place.

The mandate to use Ada applies only to software that DOD is developing and must support and maintain throughout the life cycle. Organizations are encouraged to use Commercial-Off-The-Shelf (COTS) software as development tools and even support libraries within an application. The language used for these artifacts is immaterial because DOD does not maintain the software. However, when DOD does maintain the software, the language does matter, and DOD must have an infrastructure to provide cost-effective maintenance and supportability of the software over the life cycle. Use of a single DOD language to support this infrastructure is not a new concept. DOD organizations have required the use of particular languages since the 1960s. In the mid 1970s, the use of a smaller set of languages was mandated for DOD applications. In 1985, the use of Ada was mandated for a class of DOD systems. In 1990, the Congressional mandate made Ada the language of choice for all DOD systems.

Ada is an effective language for developing high quality software to meet our DOD mission requirements at superior cost and schedule profiles over the life cycle.

1.3 EXPERIENCE OF Ada USE

Operation Desert Storm illustrates why DOD's use of a single HOL is appropriate. During this conflict, the Air Force's Theater Display Terminal (TDT) had been deployed to Israel to warn of Iraqi SCUD attacks. The TDT is a deployable missile warning system written in Ada for a SUN 3 UNIX environment. On January 11, 1991, Israel identified a new system requirement for the TDT operational program, namely, to identify the country of origin for a missile launch. Knowing the country of origin was considered important in formulating the tactical response. To Israel, it made a difference whether a SCUD was launched from Iraq or from some other

country. By January 13, 1991, an existing country-of-origin algorithm in Ada and an Ada geopolitical database were found. On January 13, 1991, these artifacts were integrated into a developmental system, and on January 14, the enhancement was integrated into an operational system. The software was flown to theater and installed for use on January 15. On January 16, only 5 days after the requirement was identified, the new capability was ready to detect Iraqi SCUD attacks on Israel. The capability to support this new mission requirement rapidly was only possible because the TDT, the country-of-origin algorithm, and the geopolitical database were in Ada. Integration of these artifacts was only possible because of the Ada package, which provided clean logical interfaces between each artifact. Had Ada not been used, the cost to support the new requirement would have been substantial, and the enhancement would not have been fielded by the end of Desert Storm.

Today, use of Ada enables DOD, Government, and commercial organizations committed to Ada to achieve a higher quality product at reduced life-cycle costs. Just as Ada use makes sense to DOD for sound, economic reasons, its use also makes sense to the commercial world for the same reasons. Digital Equipment Corporation, Boeing, Motorola, and Rockwell are some of the companies commercially selling Ada products today. Boeing's 777, the Federal Aviation Administration's Advanced Automation System, and the National Aeronautics and Space Administration's Space Station are non-DOD projects committed to the use of Ada. These organizations are openly embracing Ada, not because of the Ada mandate, but because using Ada to produce high quality software for large, safety-critical applications is a sound economic business decision.

Boeing is an excellent example of a commercial organization committed to Ada. The Boeing 777, with its estimated 10 million lines of code, will be one of the largest software projects. Boeing expects to take advantage of software reuse by reusing 2–4 million lines of existing code. Boeing also used Ada on the recent modification to the Boeing 747, which is now flown extensively by the airlines. Boeing is committed to the use of Ada for this system because of the quality, cost, and schedule benefits provided when Ada is used in combination with modern software engineering practices.

Two data points that helped Boeing with its decision to commit to Ada on the 777 was a comparison between the B-52 Offensive Avionics System (OAS) Modification in 1979–1980 and the F-22 Advanced Tactical Fighter (ATF) Demonstration Evaluation (DEMVAL) in 1990. The B-52 OAS was written in 120,000 lines of JOVIAL source code. Originally, about 15 flights had been scheduled to test the software. During flight testing, over 800 problem reports were identified, requiring approximately 80 flights to complete the software testing. The additional flight testing was an expensive cost burden. Testing was unnecessarily complicated and

knotty as, on average, 30 to 50 patches were entered to the tested software. For comparison, the F-22 ATF was written in 250,000 lines of Ada Source code. This represented far more than twice the complexity of the B-52 OAS because Ada is capable of dealing with high-level abstractions. Thirty-one flights had been scheduled to test all the complexities of the flight software. Only eight problem reports were identified against the flight software. This represents a two order of magnitude improvement from the B-52 OAS to the F-22 ATF DEMVAL. Testing was facilitated because a clean compile was available as required due to Ada's support of separate compilation. These statistics are even more impressive given that seven different contractors developed the ATF software and a team of programmers integrated it at the flight test site the week before the first flight. Certainly the advances in software engineering and tools between 1980 and 1990 played a role in these statistics. The support of these software engineering advances and tools for the Ada language was an important factor to Boeing for their commitment to Ada for the Boeing 777.

The Japanese also recognize the value of Ada in producing quality software. Covertly, Nippon (NTT) has already developed over 4 million lines of source using Ada, and several other Japanese firms are using Ada as well. The use of Ada in Japan is sensitive and closely guarded because the Japanese believe Ada provides a competitive advantage.

1.4 DOCUMENT SCOPE

This document is intended to guide Program Managers in using Ada in systems development throughout the various phases of acquisition. It also is intended to promote the use of sound software engineering principles, concepts, and processes in the system engineering process. The information contained herein should be referenced by all levels of management and technical personnel as well as by the systems development community.

The document necessarily has a strong MCCR orientation because most Ada development to date has involved mission-critical applications. However, the contents of the document should have general applicability to the AIS world as well. Program Managers responsible for developing AIS systems in Ada undoubtedly will encounter unique problems not covered by this guide. It is hoped that the references and helpful sources provided in the document will provide a useful starting point for the resolution of unique AIS issues.

The intent is to update this document annually. In keeping with the spirit and intent of the document, all users of this guide are encouraged to share their experiences and knowledge with the Ada community. Therefore, if a reader finds that significant

areas are not covered in this document, we request that he or she provide feedback to the document update process so that the next user will have better information.

1.5 CONTENT AND ORGANIZATION OF THIS DOCUMENT

This two-volume document discusses the use of Ada for software system development in the DON, and its contents apply to both AIS and MCCR communities. Volume I contains seven sections, a glossary of acronyms and abbreviations, and a reference list. Section 1 summarizes the events contributing to the adoption of Ada as the DOD programming language and provides information on the content of this document. Section 2 describes DON policy with regard to Ada use and provides a detailed policy implementation matrix. Section 3 contains specific guidance on incorporating Ada into all phases of life-cycle management—from program planning through post-deployment maintenance. In Section 4, the environments that support development and maintenance of Ada application software are described. Section 5 discusses Ada technology issues related to computers, secondary standards, the transition to Ada, life-cycle documentation, and software reuse. Section 6 presents a series of lessons learned relevant to the software development process, and Section 7 highlights what DON Program Managers can expect in the future.

Volume II contains several appendixes that provide valuable information to supplement the guidance contained in Volume I. Appendix A provides a description of additional resources that will be helpful to Ada Program Managers and users. Appendix B lists publications that may be consulted for additional information. Other appendixes amplify issues discussed in Volume I and are referenced where appropriate.

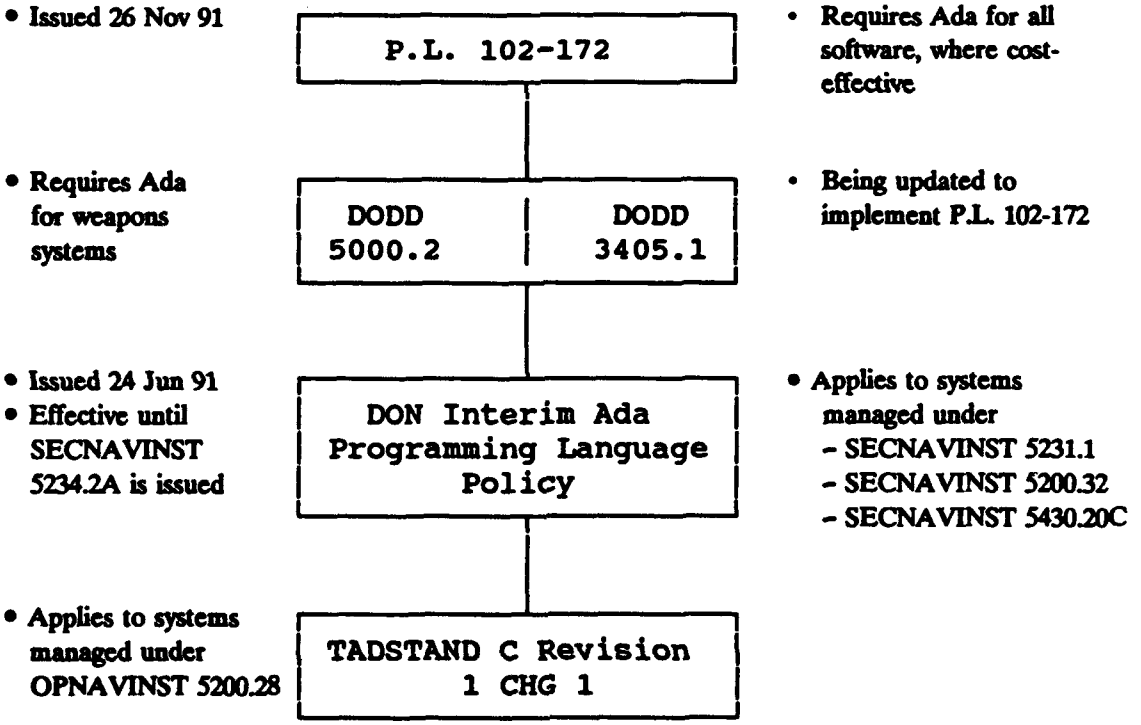
Section 2
Ada Policy

This section summarizes the Department of the Navy (DON) policy on Ada use in Automated Information Systems (AIS) and Mission-Critical Computer Resources (MCCR) programs.

2.1 POLICY DIRECTIVES

Public Law 102-172 requires that, "[n]otwithstanding any other provision of law, after June 1, 1991, where cost effective, all Department of Defense software shall be written in the programming language Ada, in the absence of special exemption by an official designated by the Secretary of Defense." Figure 2-1 provides the directives and instructions that serve as the framework for DON implementation of this law.

Figure 2-1. DON Directives and Instructions for Implementing Public Law 102-172



2.2 POLICY RATIONALE

The thrust of the Department of Defense (DOD) computer programming language policy has been to limit the number of different computer languages, including dialects and support tools, associated with the application software maintained by DOD. The Defense Department estimates that maintenance accounts for 60% to 80% of software life-cycle costs; therefore, DOD emphasis is on one High Order Language (HOL) and a limited set of standard HOLs.

The selection of Ada as the "single, common, approved standard HOL" for DOD systems was based on the inherent software engineering features of the Ada programming language. These software engineering features lead to software programs that are better structured, less error prone, and more easily maintained. In addition, these features facilitate reuse. Ada and Ada program support environments enable DOD to deal effectively with the programmatic and technical challenges posed by the increasing number and complexity of software-intensive systems.

2.3 POLICY DESCRIPTION

Compliance with Ada policy affects the selection of software resources for a system's acquisition program. Figure 2-2 describes the DON Ada policy in the context of a software resource selection matrix.

Figure 2-2. DON Ada Policy Implementation Matrix

1. Policy

AREA	Ada POLICY DESCRIPTION	COMMENTS
• Scope/ Applicability	This instruction applies to all systems and software managed under (a) SECNAVINST 5200.32, (b) SECNAVINST 5231.1B, including changes reflected in proposed SECNAVINST 5231.1C, and (c) SECNAVINST 5430.20C; all phases of the life cycles of those systems and software; and all DON components and activities, including their contractors. This instruction does not apply retroactively to the following: a. Systems that have entered production and deployment (passed Milestone III) for (a) and (b) above or systems managed under (b) above that have passed Milestone II as of 1 June 91 b. Systems managed under (a) above for which a documented language commitment was made in accordance with previous policy. This instruction will apply to systems specified in (a) and (b) at their first major system or software upgrade. This policy supersedes SECNAVINST 5234.2.	
• Thresholds	Software for use in projects at a single site that costs less than $50K in development and less than $5K/yr in maintenance does not fall under this policy. Rationale: Exempts individual PC user tasks and small office automation tasks from Ada policy.	Ada use is encouraged for these software development efforts, but they are not tracked for policy compliance below this threshold.

Figure 2-2. DON Ada Policy Implementation Matrix (continued)

Policy (continued)

AREA	Ada POLICY DESCRIPTION	COMMENTS
• Operationally Fielded Software	Conversion to Ada of existing operationally fielded software is not required. Software maintenance activity limited to error correction and modifications for portability of existing software does not fall under this policy.	Reuse or upgrade of operationally fielded software for new acquisition program or major system upgrades is addressed in Section 2 below. Reuse or upgrade of operationally fielded software for new acquisition programs or major system upgrades is addressed in Section 2 below.
• Nondeliverable Software	Nondeliverable software as defined in DOD-STD-2167A does not require Ada. Rationale: By definition, nondeliverable code will not be maintained.	
• Dedicated Processors	Ada is not required for dedicated processors that meet all of the following criteria: - Uses a maximum of 16-bit instruction set architecture - Performs a single dedicated task - Has less than 256K bytes of total memory Rationale: Allows exemptions for processors that are used for dedicated controller applications characterized by memory constraints, precluding general-purpose use.	

Figure 2-2. DON Ada Policy Implementation Matrix (continued)

2. Exceptions Requiring Written Notice From Acquisition Offices

SOFTWARE RESOURCE AREA	Ada POLICY DESCRIPTION	SPECIFIC EXEMPTION REQUEST CONTENT
• COTS Software	COTS software, including operating systems, utilities, libraries, self-contained applications programs, and vendor update implementations may be used with an exemption notification. The COTS software **may not be modified** in function or **maintained** by the Government. (The policy regarding the use of COTS software packages [e.g., DBMSs, graphics] to generate application programs that are not in Ada is addressed in Advanced Software Technology.)	Use of COTS software is encouraged and recommended. The selected software, commercial vendor, software cost or licensing arrangement, and function that COTS software performs in the system must be described.
• Reuse of Ada code	No exception/waiver is required.	Reuse of Ada software is encouraged and recommended.
• Reuse and Upgrade of Existing DOD- and Government-maintained Software	Operationally fielded software may be reused with an exception notification subject to the following conditions: - The existing source code is written in a standard HOL. - The source code modified is less than 1/3 of compilable source code. Modified code is the sum of code changes and additional code. The 1/3 change will be assessed against the smallest unit of delivery (2167-CSCI, 7935-Subsystem Specification). - Use of assembly language is identified and limited to functions required to allow the standard HOL software to run on the targeted hardware.	The reused software, function, source language, SLOC, anticipated modifications, and SSAs assigned for current and modified software must be described. DOD standard HOLs other than Ada are as follows: - ATLAS - COBOL - CMS-2 - FORTRAN - JOVIAL - Minimal BASIC - Pascal - SPL/1

Figure 2-2. DON Ada Policy Implementation Matrix (continued)

2. Exceptions Requiring Written Notice From Acquisition Offices (continued)

SOFTWARE RESOURCE AREA	Ada POLICY DESCRIPTION	SPECIFIC EXEMPTION REQUEST CONTENT
• Advanced Software Technology (AST)—DBMSs	Use of SQL (ANSI, FIPS 127-1) with compliant COTS DBMSs with binding to Ada host applications is an Ada policy-compliant approach with an exception notification. The software engineering approach to use SQL is described in: - SEI 88-MR-9 - SEI 89-SR-14 - SEI 90-TR-26	Use of SQL-compliant DBMSs is encouraged and recommended. The commercial database used, costs, and licensing arrangements must be identified. The estimated SQL SLOC and Ada application SLOC used in the application must be identified.
• AST—Graphics/ Window Environments	Development of graphics applications using existing commercial/Government (e.g., ISO, ANSI, FIPS) and Ada bindings is an Ada policy-compliant approach. - GKS Standard ANSI X3.124-1985 Ada Binding ANSI X3.124.3-1989 FIPS 120-1 ISO 8651-3 - PHIGS Ada Binding ISO/ANSI 9593-3	No waiver or exception request is required.

Figure 2-2. DON Ada Policy Implementation Matrix
(continued)

2. Exceptions Requiring Written Notice From Acquisition Offices (continued)

SOFTWARE RESOURCE AREA	Ada POLICY DESCRIPTION	SPECIFIC EXEMPTION REQUEST CONTENT
• Special-Purpose Processors	Ada is not required for special-purpose application processors (signal processors, array processors, FFT processors, etc.), provided Ada is used for the command or general-purpose processor that directs the application. Rationale: At this time, it is not cost-effective to build or use any HOL (including Ada) on processors with unusual architectures, highly restricted instruction sets, or very specialized purposes. Use of EMSP (AN/UYS-2) is included in this exception category.	
• Rapid Prototyping	Non-Ada code can be used for a rapid prototyping project. Policy requires that the project be converted to Ada before operational implementation.	The rapid prototyping effort, non-Ada language used, and Ada transition plan must be described. For MCCR systems, rapid prototyping would be part of an advance technology demonstration (6.3A) project or a project advanced under OPNAVNOTE 5000.

Figure 2-2. DON Ada Policy Implementation Matrix
(continued)

3. Ada Waiver Criteria

SOFTWARE RESOURCE AREA	Ada POLICY DESCRIPTION	SPECIFIC EXEMPTION REQUEST CONTENT
• Software Development	After all non-Ada and Ada software resources are identified and allocated to the scoping and exception categories above, the remaining software to be developed must meet the following criterion: 85% or more of the compilable source code must be in Ada. The 85% threshold will be assessed against the smallest unit of delivery (2167-CSCI, 7935-Subsystem Notification). Rationale: 15% non-Ada code is allowable as a margin for assembly-language-like functions.	For the software development effort, Ada SLOC and other non-Ada SLOC must be identified by function. Non-Ada usage beyond 15% requires a waiver. Format for waiver and exception requests is described in the DON Interim Policy.
• Research and Development (R&D)	Software resources for R&D progress in basic research and exploratory development (RDT&E funding categories 6.1 and 6.2 only) require Ada.	Such R&D projects should address applicability of Ada, the maturity of Ada tools and approaches, and Ada technology improvement efforts needed to ensure integration of target projects in the system acquisition phase. CNR is delegated waiver approval over 6.1 and 6.2 programs.

Section 3
Implementation Guidance

To realize the many benefits of Ada, the Program Manager must carefully plan for its use throughout the program's life. Department of the Navy (DON) programs have multiple stages: planning, acquisition, system engineering, software engineering, test and evaluation, and post-deployment maintenance. These stages, taken together, constitute the program life cycle. When systems engineering with Ada is used as part of this process, the Program Manager will see benefits such as reliable and high-quality software, decreased system integration problems, ease of post-deployment maintenance, and reusability of software—all of which result in reduced life-cycle costs.

This section provides guidance to help Program Managers effectively incorporate Ada and systems and software engineering into their programs.

3.1 PROGRAM PLANNING

The following key issues must be addressed during program planning to ensure that Ada is effectively incorporated into the acquisition process in the early stages:

- Organizational structure
- Resources
- Personnel and training
- Technology and tools
- Program-specific standards and procedures
- Software development process
    - Maturity framework
    - Software process assessment
    - Process control
    - Process metrics
    - Data management and analysis
- Life-cycle maintainability

The Program Manager is responsible for ensuring that planning documents thoroughly address these key issues, maintaining currency with Ada technology advances, and disseminating the planning documents. It is critical that the acquisition strategy and planning fully integrate software and hardware system development as early as possible in the process.

3.1.1 Organizational Structure

The management of an Ada software development within an acquisition program does not impose any special requirements on the Program Manager's organizational structure. Ada development and maintenance efforts are compatible with all organizational structures (i.e., functional, product, or matrix) (Archer, 1991). Using Ada does not restrict project size, functional organization, combinations of organizational structures, or the roles of the organization concerned.

As a matter of good business and system engineering practice, Program Managers, when planning their organizations, should ensure that the following conditions exist:

- Organizational structures of the Program Office, operational users, and developers (contractor or in-house developers) are integrated to support program development and/or post-deployment support. This integration can be accomplished by creating parallel organizational structures and establishing formal, informal, and electronic means of communication (Archer, 1991).

- Involved organic support organizations (e.g., laboratories, Life-Cycle Support Activities [LCSAs], Central Design Programming Activities [CDPAs], Naval Computer and Telecommunications Command [NCTC]) have clearly defined lines of authority and identified responsibilities.

- Procedures to promote the use of Ada, enforce applicable Ada policy and standards, and obtain required exceptions and waivers are in place from program outset.

- A separate staff member(s), independent from development management, should be established up front to initiate quality control planning and Independent Verification and Validation (IV&V) management. This person(s) should also be identified as the Software Engineering Institute's (SEI's) Software Engineering Process Group (SEPG) leader.

3.1.2 Resources

Modern systems are becoming more complex and frequently test the limits of the system development process. Environments procured for software development activities (see Section 4) should be robust and capable of expanding to accommodate unforeseen requirements. Therefore, development activities should evaluate competing tools and select those that are best for their application development. The current tendency of procuring host/target configurations that minimally support the requirements of the software development process must be avoided. Rather, in procuring resources, it is important to provide the software development activity with sufficient processing and memory capability to accomplish its task. In addition, when

procuring tools and environments, Program Managers must realize that adequate numbers of trained professionals will be required to use and maintain these resources.

3.1.3 Personnel and Training

The Program Manager must ensure that critical training is conducted at both management and technical levels. Project success depends on people that have the knowledge and skills needed to perform the system and software engineering, develop the process control mechanism, collect critical data and metrics, and analyze that information to determine status and trends.

Most graduates who majored in computer science and are trained in Ada and object-oriented programming methods can quickly adapt these methods for a specific application. Staff members who have spent years working with traditional programming languages and production processes may require additional training. With additional training, however, these people can become a valuable part of the program because of their inherent knowledge of corporate functions and experience in the application domain.

3.1.4 Technology and Tools

With technology advances occurring at an unprecedented rate, software and software development methods within DON must keep pace with state-of-the-art development and test technology. Keeping pace is of particular importance when considering the time required for system acquisition. The acquisition manager is responsible for ensuring that new technology is identified, exploited, accessed, and incorporated into the acquisition process when proven feasible. It is recommended that the Program Manager identify and charge a section or group within the Program Office to be responsible for the assessment of all proposed new technology. This process is important for both Government and contractor staffs. To minimize program risk, only those technologies that have been fully exercised and demonstrated should be utilized for full-scale development.

Use of the best-of-industry tools and a highly trained, competent staff ensures the highest productivity and quality. Profiles of DOD projects show that, historically, for every line of operational code, there have been at least four lines of support software (i.e., compilers, simulators, documentation tools, configuration status accounting tools) (Boehm, 1981). Traditionally, this approach has increased the hidden costs of a project in that those functions were either performed in a labor-intensive manner or additional unplanned costs were incurred in tool development. Today, most of the tools necessary for project support are Commercial Off-The-Shelf (COTS) software. However, when selecting from the diversity of tools available, the Program Manager must be aware that interface and data exchange incompatibilities can complicate

what could be a great cost benefit to the project. It is important that Program Managers direct uniformity of the support tools from the highest level of management down to the subcontractor level to ensure interface standardization.

3.1.5 Program-Specific Standards and Procedures

Documenting and disseminating the standards and procedures to manage the development process are necessary to ensure discipline and consistency in the development and maintenance phases of the software system life cycle. Documents that address the Mission-Critical Computer Resources (MCCR) standards and procedures include the following:

- Integrated Project Summary (IPS)
- Operational Concept Document (OCD)
- Acquisition Plan (AP)
- Type A Specification (Military Standard [MIL-STD]-490)
- Computer Resources Life-Cycle Management Plan (CRLCMP)
- Integrated Logistic Support Plan (ILSP)
- Test and Evaluation Master Plan (TEMP)
- Software Development Plan (SDP)
- Component-specific Engineering Notebooks (ENBs)
- Computer Resource Integrated Software Document (CRISD)
- Software Quality Assurance Plan (SQAP)
- System Configuration Management Plan (SCMP)
- System Integration Plan (SIP)
- System Engineering Management Plan (SEMP)
- DOD-STD-2167A, Defense Systems Software Development
- DOD-STD-2168, Software Quality Evaluation

Some of the above are contractor-developed documents. It is important that the content of the documents, whether developed by Government or contractor, be consistent with respect to the implemented policies and procedures.

Documents that address the Automated Information System (AIS) standards and procedures include the following:

- Mission Element Need Statement (MENS)
- System Decision Paper (SDP)
- Project Management Charter (PMC)
- Resources (RES)
- Acquisition Strategy Plan (ASP)
- Support Planning (SUP)
- Test and Evaluation Plan (TEP)

Project planning documents must identify and/or define the applicable standards and procedures, including the following:

- Use of an Ada Program Design Language (PDL)
- Documentation requirements and the specifics of how to tailor documents
- Applicable Government standards (e.g., Tactical Digital Standards [TADSTANDs])
- Program coding standards (e.g., *Ada Quality and Style Guidelines for Professional Programmers*)
- Definition and use of Software Development Folders (SDFs)
- Periodic management review of policies and procedures
- Process metrics and reporting requirements

3.1.6 Software Development Process

The basic principle of software process management is that the development process must be under statistical control. Statistical control means that valid indicators of the effectiveness of a process have been identified so that statistical monitoring of these indicators over time will identify any deviation of the process from normal operations as well as serve as a long-term basis for process improvement. A software development process that is under statistical control will, for example, produce the desired results within the anticipated limits of cost, schedule, and quality. The basic principle behind statistical control is measurement.

3.1.6.1 The Maturity Framework

Process is extremely important to any software engineering activity and has been a key research focus at the Carnegie-Mellon University Software Engineering Institute (CMU/SEI). As part of this focus, SEI has developed a maturity framework in its document titled *Characterizing the Software Process: A Maturity Framework.* (Humphrey, 1987) The paragraphs below briefly describe this framework.

The maturity framework for characterizing the status of a software process identifies five maturity levels. This process maturity structure is intended for use in conjunction with an assessment methodology and a management system. Assessment provides a way to identify the organization's specific maturity status, and the management system establishes a structure for actually implementing the priority actions needed to improve the organization.

A maturity level is a well-defined evolutionary plateau on the path toward becoming a mature software organization. Each level is a layer in the foundation for continuous process improvement. The five maturity levels in the SEI Capability Maturity Model (CMM) are defined as follows:

- **Initial**—This lowest level of process maturity refers to an ad hoc software development process. Until the process is under statistical control, orderly progress in process improvement is impossible.

- **Repeatable**—A stable process with a repeatable level of statistical control is achieved by initiating rigorous project management of commitments, cost, schedule, and change.

- **Defined**—Definition of the process is necessary to ensure consistent implementation and to provide a basis for better understanding of the process. At this point, it is probable that advanced technology can be usefully introduced.

- **Managed**—Following the defined process, it is possible to initiate process measurements. At this level, the most significant quality improvements begin to appear.

- **Optimized**—With a measured process, the foundation is in place for continuing improvement and optimization of the process.

If an organization is at the Initial Level (Level 1), it must achieve a *disciplined process* to advance to the Repeatable Level (Level 2). The key areas associated with this disciplined process are as follows:

- Software configuration management
- Software quality assurance
- Software subcontract management
- Software project tracking and oversight
- Software project planning
- Requirements management

Appendix C provides additional information on how to advance from a Level 1 to a Level 2 organization. Organizations would benefit from a formal assessment that ascertains what is necessary to advance to Level 2 (the Repeatable Level). Attainment of this level is important because it enables management of commitments, cost, schedule, and change. Assessment of contractors who have submitted proposals in response to Requests for Proposals (RFPs) also would be

beneficial to ensure that the contractor organizations are capable of implementing their proposals.

3.1.6.2 Software Process Assessment

The DON recommends that projects assess the maturity level for contractors and in-house software development before beginning software acquisition or development.

A five-level software process maturity model has been formulated. The hypothesis is that an organization's process maturity level will be an indicator of the extent to which an organization's software process can be depended upon to produce and sustain high-quality software in a predictable, controlled way. A software process maturity questionnaire has been developed for use in two different SEI-developed methods (i.e., Software Process Assessment [SPA] and Software Capability Evaluation [SCE]) for determining a software organization's maturity level.

The SPA method is a self-diagnostic and improvement-initiating activity that helps software organizations launch effective process improvement programs. SEI or its nine licensed vendors provide SPA training. SPA produces a baseline of the organization's process maturity based on examination of five or six representative projects. These projects may be old or new, large or small. They may be using either the latest languages and methods or established languages and methods.

The SCE method is used by Government acquisition organizations to help identify the most capable software organizations during source selection and to monitor contractor capability during contract performance. The SCE method is an audit technique. SCE examines the organization in light of the contract to be awarded. The software organization chooses eight projects from which four are selected for evaluation by the acquisition organization as an indicator of its ability to perform on the contract on which it is bidding.

3.1.6.3 Process Control

Program Managers must ensure that reporting mechanisms and procedures are in place to regularly review the development status. They also must ensure that a method exists for assessing both the development process adherence to plans and risk. The resolution of critical problems must include inputs from the end user, Program Managers, and developers.

Standardization of the metrics process is critical to allow analysis that will reveal the cause rather than the side effects of the problem. Formal recordkeeping and accurate traceability of top-level requirements down through code and unit test and test bed development and execution are vital to the analysis of system problems.

Program Managers should institute a SEPG as the primary means of process improvement and quality control. This should also be required of contractors.

3.1.6.4 Process Metrics

Software metrics refer to the measurement of pertinent software processes and product parameters that support defined requirements. Integrating metrics into the development process will enable the objective and continuous assessment of development progress, resource expenditure, and technical product quality. Metrics must be identified, qualified, and applied at the beginning of the acquisition and system engineering processes.

For each new program, the Program Manager must ensure that, before contract award, a software metrics implementation plan is in place that provides for the definition and collection of metrics, data analysis, and interpretation of results. An experienced support organization in the Government must be identified to implement an effective metrics program. This organization can initiate assessment activities during source selection by working with RFP-specified and contractor-proposed software planning metrics estimates. The implementation program should adhere to the following guidelines:

- **Metrics should address specific issues of interest to the Program Manager.** Many issues are common across programs (e.g., cost/schedule, personnel resources, Source Lines of Code [SLOC], and defects). Therefore, a standard set of metrics applicable throughout major acquisition programs should be identified. The metrics program should start small, using a core set of metrics. The program should be flexible and expandable to address additional issues that may be derived from the analysis of the core set or other program concerns. These additional issues will depend on program size; acquisition strategy; the developer's process; cost, schedule, and technical risks; and changes in these characteristics throughout the life cycle.

- **Metrics should be applied throughout the life cycle.** Because metrics are linked to a software process or product, metrics need to be collected on a planned basis throughout the system life cycle.

- **Metrics should be defined and used consistently.** Consistent definitions provide a critical foundation for meaningful comparisons (e.g., plans versus actual, between configuration items).

- **Metrics should be collected automatically when possible to ensure accuracy, consistency, and timeliness.** Data should be accurate, current, and consistent

and should be time-stamped to record timeliness. On-line access to developer data in electronic format should be provided throughout the life cycle. Developer data should be treated as proprietary to prevent misuse and reduced availability.

- **Metrics should be analyzed and interpreted to assess achievement of milestones and quality of software products.** Metrics can be used to identify and assess issues. Raw metrics can be misleading; to be useful, they must be interpreted with respect to the overall software process.

- **Metrics assessment information must be transferred to and used by decision-makers.** The quantitative results of software metrics analysis can be used as an indication of the overall health of the program or project and give normalizing parametrics to the overall reporting process.

- **The definition of the SLOC should be consistent for all of the DON activities.** The recommended definition is provided in Appendix D of Volume II.

To establish an effective metrics program, the requirement for metrics must be formally established in contractual documents (e.g., RFP, Statement of Work [SOW]). Metrics requested in these documents should be tailored for the application. As mentioned above, metrics should be defined and used consistently within all activities using that metric. One such definition that should be common across all DON activities is the SLOC definition (see Appendix D). This definition has been used effectively within the Naval Air Systems Command (NAVAIR) and allows for comparisons between Ada and non-Ada programs. Appendix E provides an example of wording for use in a contractual document to effect a metrics program. This example is also based on NAVAIR use and should be tailored to your Software Engineering Environment. Your application development may require additional metrics.

Other useful metric information can be found in the following documents:

- San Antonio I, Panel I, *Software Metrics Implementation Final Report*

- U.S. Air Force, *Air Force Systems Command Software Quality Indicators: Management Quality Oversight*

- U.S. Air Force, *Air Force Systems Command Software Management Indicators: Management Insight*

- National Aeronautics and Space Administration (NASA), *Software Engineering Laboratory (SEL) Guidebook*
- Humphrey, W.S., *Managing the Software Process* (CMU/SEI)
- CMU/SEI, *Software Metrics*

The Air Force document on software quality indicators mentioned above addresses acquisition concerns and provides an excellent approach to metrics. The technical report prepared for the Naval Air Development Center (see Dyson, 1989) describes how various indicators can be used for a project.

3.1.6.5 Data Management and Analysis

Databases are used for storing documentation, source code, test materials, financial data, personnel data, correspondence, problem report status, and the like. To enable these data to be integrated into a project-wide picture for management analysis, at a minimum, the underlying database formats must have interface consistency. The Program Manager should establish interface design documents based on standard database dictionaries.

3.1.7 Life-Cycle Maintainability

Writing software in Ada does not guarantee ease of maintenance. The capability to quickly isolate and correct faults must be designed into the system. The capability to make minor enhancements without redesigning the system also must be incorporated into the design.

Program Managers should use an in-house organization to monitor the software development. The in-house organization participates in software technical reviews, performs IV&V activities, and participates in formal testing. If the in-house organization is the future Software Support Activity (SSA), the project can be guided to meet its requirements for maintainability. This will simplify the transition of responsibilities and lower the long-term costs. Poorly structured or inflexibly designed software will require extensive efforts for minor changes, thus driving up the life-cycle costs.

3.2 ACQUISITION PLANNING

Acquiring contractor support for a software system requires the preparation of many documents. To ensure that a competent Ada contractor with extensive system engineering experience is selected, the following documents, in addition to the ones listed in Section 3.1.5, can be used:

- Acquisition Plan
- Statement of Work (part of RFP)
- Proposal preparation instructions (part of RFP)
- Proposal evaluation criteria (part of RFP)
- Government estimate
- Deliverables—Contract Data Requirements List (CDRL)

3.2.1 Acquisition Plan

Acquisition planning is critical to the success of any program. From the software perspective, the following requirements should be included in the AP:

- Use of Ada
- Application of software engineering principles
- Demonstration of contractor competence in Ada and software engineering process or a plan to reach competence if contractor was selected on other factors
- Risk management

3.2.2 Statement of Work

Throughout the Work Breakdown Structure (WBS), Ada use should be specified. It should be clear where further analysis may be required to determine whether a waiver should be applied and where there is no question about Ada use. In addition, those elements that are subject to Ada reuse from the Government-Furnished Equipment (GFE) and other domains would be specified. Other issues to be addressed would include:

- Project application of SLOC definitions (see Section 3.1.6 and Appendixes D and E)
- Definition of all categories of software and firmware
- Applicable TADSTAND requirements for MCCR
- Estimated software sizes by configuration items
- Scope of compliance to applicable standards (e.g., DOD-STD-2167A and Navy Data Automation Command Publications [NAVDAC PUBS] 24.1 and 24.2)
- Software metric requirements

- Requirements of development plan, configuration management plan, quality assurance plan, and life-cycle management
- Integrated automated documentation
- Use and delivery of test software, support software, and project-specific data files

3.2.3 Proposal Preparation Instructions

It is recommended that, as part of the RFP package, bidders be required to submit draft planning documents and respond to the following issues:

- Commitment to Ada-oriented system and software engineering processes
- Demonstrated corporate adoption of Ada (e.g., Ada training, Ada experience)
- Procedures to coordinate with the Federal Acquisition Regulations (FAR) for reuse of a corporate and/or domain-specific Ada repository
- Statements regarding rights/data issue (i.e., data rights escrows, documentation, quality and testing, software liabilities, contract incentives) concerning proprietary Ada technologies
- Procedures for enforcing the tenets of Ada and software engineering in the subcontractor environments
- Software development process control
- Software metrics program
- Performance of initial allocation of function to configuration items with specification of language and estimated SLOC
- Summary of successful performance on previous programs requiring the use of Ada

3.2.4 Proposal Evaluation Criteria

The RFP evaluation criteria should include the following:

- Requirements traceability
- Cost modeling

- Personnel experienced in Ada
- Plan for acquiring the required Ada-capable staff needed to meet contract requirements
- Corporate and/or domain-specific Ada reuse library
- Visible Ada training program
- Ada projects completed and size (SLOC)
- Adoption of Ada for internal project use
- Software process maturity level of repeatable or higher
- Controls and procedures that enforce subcontractor compliance with the proposed evaluation criteria
- Validation that software sizing, cost, and schedule are within Government estimates
- Tools/environments applications
- Testing process
- Risk assessment
- Risk management
- Quality assurance plan
- Active SEPG

3.2.5 Government Estimate

In developing the Government estimate, the following issues should be considered:

- Experienced Ada programmers tend to be more expensive.
- Contractors who have not completed two or three Ada projects will be less productive than more experienced contractors.
- Developing code to be reused on other projects is more expensive.

- To achieve maximum benefit, Computer-Aided Software Engineering (CASE) tools must be integrated into the software development process.

- Labor-hour totals and category allocations should be estimated for a worst-case scenario.

- Reuse of existing Ada code may lower project costs and result in a higher quality product.

- Use of complementary open standards (e.g., XWindows, MOTIF) can significantly reduce cost through improved productivity.

3.2.6 Deliverables—Contract Data Requirements List

After the foregoing parts of the acquisition process have been completed, the deliverable products must be described. The CDRL specifies which deliverables will be required and the quality that will be expected. Close coordination in preparing the SOW and the CDRLs should result in very high quality software deliverables. The CDRL enables the Program Manager to describe, tailor, and specify, by means of the Data Item Descriptions (DIDs), the nature, detail, and the "personality" of the software deliverables (What did I really get as a product?). The CDRL is used to specify the product review cycle and pattern (How many times and to what level of detail can I review each product during the development?). In addition, the CDRL is used to specify the number of products into which to divide the "software" to ensure understandability, testability, and life-cycle supportability (How do I know what is in that "computer full of software?"). BLOCK 16 of the CDRL is provided to allow the statement of additional requirements that will ensure the quality of a product.

A detailed CDRL does carry some economic impact. Product quality, however, reduces the life-cycle cost of ownership. Bidders should be instructed to assess the CDRL specifications carefully, and evaluators should be extremely critical of these issues. The CDRL serves as an "insurance policy" for good systems engineering to yield quality products.

The Program Manager should specify that deliverables be created and maintained electronically in accordance with Computer-Aided Logistics Support (CALS).

3.3 SYSTEM ENGINEERING AND RISK MANAGEMENT

System engineering should take a programmatic view that recognizes software costs as a significant portion of the total system life-cycle cost. Consequently, software must be considered in the early stages of a project.

During the system definition phase, the Program Manager should perform analyses to identify high-risk requirements and potential solutions in the context of both hardware and software. High-risk hardware and software components should remain visible throughout the development process. Areas of risk management for Ada implementation include the following:

- Software versus hardware in a system context
- Prototyping
- Project context benchmarks
- Requirements volatility and traceability
- Support software acquisition impacts
- Ada software reuse
- Prime contractor-subcontractor relationships
- Testing philosophy and methodology
- Integration philosophy
- Incremental development

3.3.1 Software Versus Hardware in a System Context

Software is the single largest cost growth area in the defense industry; therefore, the Program Manager should focus on software. The concept of "software first" means postponing hardware selection until the software design is completed. Implementation of this concept allows the Program Manager to select hardware capable of supporting the software requirements. The software requirements should be defined in parallel with hardware requirements as a function of sound system engineering practice. Complex software solutions to resolve hardware limitations lead to expensive development and life-cycle impacts. Actual code production should be deferred until requirements allocation and system design have been completed and baselined.

3.3.2 Prototyping

Prototyping is used to evaluate alternative algorithms and confirm requirements. It is recommended as a method of risk abatement and not as a means to rapid deployment. Fielding a prototype system can lead to life-cycle cost impacts that far exceed the theoretical development cost savings. Prototyping also can be a repetitive spiral of alternative solutions that gradually narrows the choices to single out the preferred approach. (Software prototyping does not mean hacking.) Each alternative must be planned for and accompanied by disciplined documentation of the design approach and system interfaces. Prototyping can help validate system requirement and design correctness.

3.3.3 Project Context Benchmarks

Development is recommended of benchmark programs that represent the critical aspects of a software application (e.g., Kalman filters, operating system overhead, track correlation algorithms, graphic output) to determine whether available compilers and/or code generators satisfy projected operational requirements. The benchmarks also help to identify quickly the hardware deficiencies in the project (see Section 5.2.3 for more discussion of benchmarks).

3.3.4 Requirements Volatility and Traceability

Requirements volatility increases costs, schedule, and fielded error rates. To gain the maximum productivity from Ada, the software requirements should be baselined at the Software Requirements Review (SRR), but certainly no later than at the Preliminary Design Review (PDR). At PDR, the allocation of software function to Computer Software Configuration Items (CSCI) components should be complete. At Critical Design Review (CDR), allocations of these functions to respective Ada package specification should be complete. Any change of requirements beyond this point could cause the functional allocation of requirements to be "rearchitected" to their respective package specification. Rearchitecting the functional allocation requirements will affect the schedule, whereas folding the requirement changes into the existing structure could affect development communications overhead, (i.e., cost and schedule). Rearchitecting also has the side effect of invalidating baseline documentation. The effort required to fix the affected documents has to be considered. Requirements volatility can be reduced through prototyping, configuration management, change control, and design revision. The configuration management method should support traceability of the requirements to the design as well as from the design back to the requirements.

3.3.5 Support Software Acquisition Impacts

When using COTS software, the Program Manager must consider the stability of the product vendor to ensure that the support environment will be available for the life of the system. The identification of data and data rights must be part of the acquisition of any COTS software. Appendix B lists several SEI publications on data rights that are available from either the Defense Technology Information Center or SEI.

3.3.6 Ada Software Reuse

Software reuse is important to the DOD and the Ada community because reuse promises a higher quality product with significant cost savings across the software life cycle. Although software reuse was not specifically identified as one of the design criteria for the Ada language, the concept of reuse is strongly implied by the design criteria.

Major reuse repositories are NASA's AdaNet, the Ada Software Repository (ASR), the Common Ada Missile Components (CAMP) effort, Reusable Ada Products for Information Systems Development (RAPID), and the Software Technology for Adaptable, Reliable Systems (STARS) repository.

In addition to these repositories for public-domain software, a commercial market of reusable software is emerging (e.g., Grady Booch's Software Components [Booch, 1987] and EVB's Generic Reusable Ada Components for Engineering (GRACE™) [EVB, 1987]). Two versions of the videotape on Ada reuse, *Common Ada Missile Packages—Leading the Way in Software Reuse*, were developed on the CAMP-3 effort. One version runs 20 minutes, and the condensed version runs 10 minutes. The videotape provides an overview of Ada, software reuse, and the CAMP program. (See reference list entry, McDonnell Douglas Missile Systems Company, 1991.)

Domain analysis can be performed as part of the Requirements Analysis Phase. The domain analysis will identify the commonalities and variations of the existing systems within the same application area. The expected benefits are a better understanding of the existing systems in terms of common functionality and identification of components from existing systems that can be reused in the current development. An additional benefit from domain analysis activity is to help stabilize requirements.

Use of Ada as the implementation language does not, by itself, ensure reusability. To ensure reusability, it is necessary to have a library of reusable programs as well as to establish Ada programming standards, policies, and procedures. Libraries of reusable Ada programs have great promise for reducing future software development costs. The Program Manager must be sensitive to what will make maintenance of the library attractive to the development contractor (i.e., data rights). Section 5.4 provides additional details on software reusability.

3.3.7 Prime Contractor-Subcontractor Relationships

In selecting subcontractors, the prime contractor should apply the Ada technology acquisition criteria discussed throughout Section 3.2. The prime contractor must be as analytical in selecting its subcontractors as the Government is in selecting the prime contractor. The Government should be as aware of the subcontractor's capabilities and commitment to Ada as it is of the prime contractor's. The relationship between the prime contractor and its subcontractors should not be taken for granted. The RFP should specify that the contractor-subcontractor relationship be described in the proposal and should include proposal evaluation criteria that address this relationship.

3.3.8 Testing Philosophy and Methodology

Because of the increasing complexity and size of emerging systems, testing philosophy must be considered and built in at the concept formulation phase of a project. All requirements, developed and derived, must be evaluated not only for their application to the problem statement or mission, but also for testability. Large-scale, heterogeneous, distributed systems can no longer be tested while the hardware and executable code are being "put together" in the development cycle. Testing must start in the requirements definition phase (e.g., requirements validation, requirements realism, transitioning of requirements to function). Systems must be specified to ensure and enforce requirements for unit testing of the individual functional capabilities; full-scale integration testing; verification, validation, and certification testing; and life-cycle support testing.

The testing methodologies include use of designed-in test structures and application of tools for modeling, simulation, design, development, and operational assessment of systems in both the hardware and software arenas. Ada lends itself to very structured, defined testing when used as part of a well-defined system engineering policy.

3.3.9 Integration Philosophy

Navy systems are becoming so large and complex that systems integration is growing in proportion and importance. Both the tactical and nontactical worlds are developing highly complex, multifunctional systems, and the distinctions that once existed between the systems used in these two worlds are becoming blurred. A classic example is the critical time requirements of modern logistics systems to support the rapid deployment, and multimission functionality of today's Navy forces.

Ada use increases in importance in light of the diversity of hardware required for the wide variety of these operational environments. Ada can be applied over most of these environments for mission performance and problem solving. With the appropriate systems engineering practices and policies, Ada could contribute significantly to easing the multivendor, multimission integration problems encountered in today's systems.

3.3.10 Incremental Development

The cost of developing systems has increased beyond our imagination (and budgetary constraints). Of necessity, therefore, future systems development will be evolutionary; that is, systems will be developed incrementally. Immediate need and affordability will set the priorities for the problems to be solved and the requirements and functions to solve these problems. Again, systems engineering, policies, and Ada will play major roles in this evolutionary development. Although the life cycle of a major system or collection of systems may span 15 years (four to eight generations

of hardware) during which operational requirements will change, Ada will support the life cycle and technology infusion of these systems. With careful and reasonable planning, Ada will support the application of new technology and changing requirements better than other programming languages have done because Ada was developed with reusability, transportability, hardware independence, and upgradability and maintenance of software as important issues.

From a long-range development planning perspective, incremental development is a "build-a-little, use-a-little" cycle. This effectively places a system in a long-range upgrade life cycle. In fact, what seems like maintenance is really development (today's system maintenance, which upgrades rather than "fixes," is also "development").

3.4 SOFTWARE ENGINEERING

Good system engineering practices provide a framework for good software engineering practices.

3.4.1 Ada Software Engineering Goals

Grady Booch (1983) defined the discipline of software engineering as follows:

> The underlying cause of the software crisis is that software systems have become unmanageably complex. Furthermore, we cannot expect them to become any less complex, for as we improve our tools and gain experience in designing software systems, we actually open up even larger, even more complex, problem domains.
>
> As a solution to this crisis, we must therefore apply a disciplined artistry, using tools that help manage this complexity. In a broad sense, we call this discipline software engineering.

The discipline of software engineering has identified four goals that are supported by seven software engineering principles to help manage the complexity of developing modern software applications. The four software engineering goals are understandability, reliability, modifiability, and efficiency, which are defined as follows:

- **Understandability** is a key software engineering goal for the management of complexity. For a system to be understandable, it must reflect our natural view of the world. At the high level, objects and operations map to real-world data and algorithms. At the low level, the software solution is understandable as a result of proper coding style.

- **Reliability** is associated with the quality of the software and is a critical goal where the cost of failure is high. Reliability issues must be addressed throughout the design process. Reliability can be built only from the start; it cannot be added at the end.

- **Modifiability** deals with the capability to perform maintenance on or otherwise change the software. A change in requirements should result in a controlled change in the software. Error correction to the software should be effected as a controlled change to the software.

- **Efficiency** refers to the use of resources. Time and space resources should be used optimally. This goal is especially important when real-time deadlines must be met to satisfy the application requirements.

3.4.2 Software Engineering Principles

The seven principles of software engineering that support the goals of software engineering are abstraction, information hiding, modularity, localization, uniformity, completeness, and confirmability.

- **Abstraction** allows users to highlight the essential details of a process or its data dependencies and omit the unessential details. In this manner, the logic of a program solution can be expressed in terminology approximating the problem domain rather than in computer-dependent terms. Abstraction supports code readability and maintainability. The essential details can be filled in later and/or reworked without impact on the balance of the system.

- **Information Hiding** makes inaccessible certain details that should not affect other parts of a system. For example, a disk drive should be controlled as a collection of files. An application should not control a disk drive by using tracks and sectors; doing so could violate data integrity concepts implemented as part of another process. Reliability of systems is enhanced when, at each level of abstraction, only certain operations are permitted, and any operations that violate our logical view of that level are prevented.

- **Modularity** organizes very large programs into discrete parts, allowing separate development of the individual components. This concept is supported by the concepts of package specifications and package bodies. Top-level design uses the package specifications to define the architecture of the discrete components and their data interrelationships. This structuring through the package specifications creates templates that guide the program elaboration (coding of

the package bodies) and incremental build and test schedules. The principle of modularity directly supports the goals of modifiability, reliability, and understandability.

- **Localization** creates programs in which each part is highly cohesive (i.e., critical data are self-contained) and loosely coupled (i.e., a part can execute in isolation). Localization enables development of self-sufficient components that can be implemented with minimal technical inter- and intraproject communication. Modularity and localization are key components in reducing expensive project communications overhead and critical to the incremental build and test. The principle of localization directly supports the goals of modifiability, reliability, and understandability.

- **Uniformity** uses a consistent notation for all artifacts within a software development activity. Modules are free from any unnecessary differences. The principle of uniformity results from a good coding style and directly supports the goals of modifiability, reliability, and understandability.

- **Completeness** creates programs that completely satisfy both behavioral and performance software requirements. Completeness helps us develop solutions that are correct by ensuring all of the important elements are present. The principle of completeness supports the goals of modifiability, reliability, understandability, and efficiency.

- **Confirmability** verifies that the application software developed satisfies all requirements. Each software system must be readily tested. Decomposed systems can localize testing, thus helping to make our systems modifiable. Ada's strong typing facilities help the confirmation process. Specialized automated tools can also support the confirmation process. The principle of confirmability supports the goals of modifiability, reliability, understandability, and efficiency.

The Ada language was developed to support software engineering goals through features that draw on these software engineering principles. If these principles are understood thoroughly and applied on an Ada project, use of Ada can effectively support the software engineering goals. Using Ada without knowledge of these principles will not automatically result in the generation of good code.

3.4.3 Ada Adaptation Guidelines

Ada was designed to support the development of large special-purpose programs that have an extended life cycle. Typically, the projects are accomplished by a

programming staff divided among multiple contractors and/or corporate entities. The following recommendations are intended to help such projects incorporate Ada and sound software engineering practices into their development process:

- Use macro-economic cost-estimating tools to help develop cost and schedule estimates and to allocate the time and resources to the respective development phases. Examples of such tools include COCOMO, PRICE-S, SASET, REVIC, SOFTCOST, and SLIM. The Navy Center for Cost Analysis can provide more information and limited training and support as needed.

- Beware of overloading the project with excessive documentation requirements by defining too many configuration items. Conversely, too few requirements may cause loss of visibility and control. Allocating operational functional requirements to configuration items should be both a management and a technical decision because it establishes the framework for collecting information on software requirements, design, code, and testing.

- Use the DOD-STD-2167A tailoring package to ensure that Ada design aspects are maintained while trying to conform to a strict interpretation of DID requirements. Design DIDs should be tailored by experienced Ada design technicians. For additional information on the use of Ada on DOD-STD-2167A programs, see *Implementing the DOD-STD-2167 and DOD-STD-2167A Software Organizational Structure in Ada.* (Association for Computing Machinery [ACM] Special Interest Group on Ada [SIGAda], 1990).

- Select a software development model appropriate for the project applications. Although DOD-STD-2167A defines the waterfall development process, other models may be used (e.g., the Spiral Software Development Model or the Ada Process Model). Additionally, it is expected that DOD-STD-2167A will be tailored to meet each project's specific needs.

- Allocate work to organizations whose management, personnel, and technology support the highest available maturity level. CMU/SEI has established assessment materials that will help define the software engineering maturity level of a development organization. For more information on the maturity levels, see *A Method for Assessing the Software Engineering Capability of Contractors* (Humphrey and Sweet, 1987).

- Ensure that the metrics employed include a clear definition of the component parts (i.e., SLOC), are accurate and readily collectible, and span the development spectrum and functional activities. The Naval Sea Systems Command (NAVSEA) Software Quality Initiative provides an example of a program with good software metrics.

- Ensure that compiler-checked Ada package specifications are used as the focal point for the PDR.

- Use Ada's separate compilation features to provide direct support for program configuration management and process metric collection.

- Before proceeding from the PDR milestone, ensure that the top-level design has been iterated in package-specification form so as to eliminate all major risk items. Any volatility in requirements or technical redirection after PDR that affect the design will place at risk the cost-saving advantages of a disciplined engineering process.

- Establish a set of rigorous programming standards to ensure that readability is built into the Ada source code to maximize Ada's self-documenting features.

- Establish an incremental build process, focused on the package specifications, so that the system evolves from its kernel to full capability. Schedule run time and shared services test and/or demonstration early in the development cycle.

- Use databases and documentation tools to support the software life-cycle management process (i.e., design, coding, testing, and support). An automated SDF database supports the detailed design (elaboration of the package body in PDL), the coding (elaboration of the PDL representation to full Ada source), and unit testing. The SDFs become the focal point for the CDR (Boehm, 1989). The SDF database should be a contract deliverable and should be maintained in an electronic database format for future delivery to the SSA. Maintaining SDF data in machine-readable form will increase accessibility for Government visibility. An approach to an SDF database is provided in *TACAMO Software Development Files (SDF) Definition.*

- Use Ada's portability by promoting code and unit testing component parts while integration is conducted elsewhere. For example, conduct code and unit tests on a personal computer (PC), then move the build test to a mainframe and integrate/test in a mockup.

- Identify all standard interfaces (e.g., Portable Operating System Interface for UNIX [POSIX]) to be used in the project during the requirements phase. This effort is critical to portability, reusability, and accommodation to evolving technology. The standard Ada bindings should be specified in all documentation affecting the software development process.

- During the requirements phase, thoroughly search all Ada repositories for candidate routines for inclusion in the developing system. The reuse capabilities of Ada, in conjunction with the repositories, provide an opportunity for considerable savings in both time and money.

3.5 TEST AND EVALUATION

Product development and its associated test-bed development are parallel efforts. Use of parametric modeling (e.g., Ada COCOMO, PRICE-S, REVIC) allows the test and integration phase to be defined in terms of time and staff resource requirements. When using any cost-estimating tool, it is important to be objective in setting the cost factors. It is recommended that a conservative approach be used and that estimating tools be one of several estimate sources. The cycle of incremental builds, test, and repair should be a well-defined and configuration-managed process. If the software engineering process objectives outlined in Section 3.4 have been followed, Ada's strong type-checking, compiler-error-checking, and self-documenting style should reduce errors.

The later in the process an error is discovered, the greater the cost to correct it. Within the spectrum of product development, management must stress processes that will minimize errors before the integration and test phase. The cost of reducing errors at design time is significantly less than the cost to repair during static unit test, which is significantly less than the cost to repair during integration and test. Error repair should focus on source correction rather than patching. Converting patches to source codes represents a redundant process. The cost impact of error repair during integration testing demonstrates that this phase should focus on integration, not discovery of unit code errors, bad design, or ambiguous requirements.

3.6 POST-DEPLOYMENT MAINTENANCE

The In-Service Engineering Activity (ISEA) and LCSA efforts require the same engineering disciplines that are applied to the initial development of the operational system. Life-cycle support consists of a repetitive cycle of development projects. The common denominator is that each cycle builds on the results of the previous cycle. In the past, after a few revision cycles, FORTRAN, JOVIAL, or Compiler Monitor System 2 (CMS-2) implementations typically have resulted in significant degradation of the supporting design documentation, the quality of embedded comments, and the integrity of the system architecture. The combination of well-defined programming

standards and Ada's syntactic support for self-documentation provides a means to alleviate this problem. In addition, automated documentation tools can reduce cost and provide a vehicle for timely update. The software engineering goals described in Section 3.4 are the cornerstones for ensuring long-term, cost-effective maintainability and ease of revision of the supported operational program software.

Unique to the life-cycle effort is the requirement for life-cycle support staff who conduct on-site visits to diagnose problems in an operational environment. Traditionally, a few specialized systems personnel who can work at a machine-dependent level of detail have performed this activity. These personnel have relied more on their system knowledge than on support tools.

The Program Office should ensure that test analysis tools are built to allow life-cycle support staff, known as a contact team, to work at the highest possible level of abstraction, not at the machine level. Symbolic debug capabilities magnify the capabilities of the individual team member. The most expensive debugging occurs in the operational environment, and the most effective way to reduce the expense is to eliminate the need.

3.7 HIGHLIGHTS

The preceding subsections provide detailed guidance for Program Managers in incorporating Ada and software engineering into every phase of a program's life cycle. This subsection highlights the critical points in each phase.

Guidance for the Program Planning Phase, for example, emphasizes the need for integration, up-front preparation of procedures, and careful tool analysis and selection. This guidance may be summarized as follows:

- Integrate software and hardware system development as early as possible.
- Ensure that the program organizational structure(s) integrates the activities of the Program Office, the operational users, and developers; embodies clearly defined lines of authority; and supports promotion of Ada use and enforcement of Ada policy and standards.
- Identify, document, and disseminate standards and procedures needed to manage the development process.
- Identify metrics to be applied at the beginning of the system engineering and software implementation processes.
- Develop a software metrics implementation plan before contract award.

- Ensure that the development process is under statistical control from the outset.

- Before selecting development tools, analyze carefully their processing and memory capabilities to ensure they meet program needs and choose best-of-industry tools.

- Analyze new technologies carefully and use only those that have been exercised and demonstrated.

- Plan for training.

Guidance for the Acquisition Planning Phase relates primarily to the requirements of the RFP. Highlights of this guidance are as follows:

- Develop a strong acquisition plan and other documents needed to ensure selection of a competent Ada contractor.

- Ensure that the RFP SOW provides specifications on Ada use and/or reuse, defines all of the applicable software and firmware categories, specifies conformance to applicable standards, and states the software metrics requirements.

- In the RFP evaluation criteria, emphasize contractor experience, training, and competence in Ada use and reuse as well as requirements traceability, cost modeling, and subcontractor compliance with the evaluation criteria.

For the System Engineering and Risk Management Phase, the guidance provided stresses the importance of considering software-related issues from the outset and identifying high-risk hardware and software components. Specific guidance in this area includes the following:

- Define software requirements in parallel with hardware requirements.
- Use prototyping and benchmarks to mitigate risk.
- Freeze requirements no later than the PDR.
- Identify opportunities for and consider reuse of Ada source code.

Software Engineering guidance focuses on applying good software engineering practices while meeting Ada goals. Included in this guidance are the following:

- Develop software for the program that meets the Ada goals of abstraction, modularity, localization, maintainability, reliability, and reusability.

- Follow good software engineering practices.

- Implement the recommendations proposed in Section 3.4.3 for developing a model for software development.

The guidance for the Test and Evaluation Phase is based on the premise that product development and its associated test bed development are parallel efforts. Specific guidance in this area includes the following:

- Implement processes that identify errors early in the development phase to avoid costly repair at the integration and test phase.

- Use parametric modeling and cost-estimating tools.

- Ensure that the cycle of incremental builds, test, and repair are under configuration management.

With respect to the Post-Deployment Maintenance Phase, the guidance emphasizes the similarity between the engineering principles used in initial development of the operational system and post-deployment maintenance activities. To reduce the expense associated with debugging in the operational environment, the guidance proposes building test analysis tools that allow a contact team to work at the highest possible level of abstraction.

Section 4
Environments

This section discusses the environments used to support the development and maintenance of Ada application software.

4.1 SOFTWARE ENGINEERING ENVIRONMENT

The term "environment" evolved from early work on the UNIX operating system. It originally referred to setting certain parameters and characteristics of the operating system to create a programming or computing environment suitable for the user. From the start, the term implied selecting or tailoring the environment to meet specific user needs. Gradually, the term became more general, referring to Programming or Project Support Environments (PSEs), thus reflecting the idea that the environment was designed and constructed to support a particular class of applications. The Software Engineering Environment (SEE) became a set of computer tools to support the activities of software engineering.

A SEE that supports software development is a computer-based set of integrated methods, tools, and procedures to develop software that meets mission needs. This definition inherently means that a SEE must provide the functions of both a programming development system and a management information system to monitor and control the development. The term "Ada environment" means that the scope of the SEE is focused on supporting the development of software written in the Ada programming language and the associated software engineering methods and procedures.

Currently, the accepted practice is to execute the SEE on commercial host computers to develop project software during the requirements, design, coding, integration, and testing phases. Normally, software testing and system integration and testing are performed in hot-mockup test facilities with support from the SEE. The SEE is then used for the remainder of the system life cycle.

Other terms for SEEs that appear in the general literature include Software Development Environment (SDE), Integrated Software Engineering Environment (ISEE), Project Support Environment (PSE), Ada Programming Support Environment (Ada PSE), and Integrated Project Support Environment (IPSE). A PSE is a SEE that has been configured to meet the needs of a specific project; the term PSE, which appears in the remainder of this section, is used interchangeably to mean project and programming support environments.

4.2 TYPICAL TOOL SET

A few tools are used in almost every application although several other tools exist that are desirable and can improve productivity across the software life cycle. The typical tool set, however, consists primarily of those tools associated with the coding phase of development, including the following:

- Editor—Used to create or modify source and to view or modify files produced by other tools
- Compiler—Translates a High Order Language (HOL) source program into its relocatable code equivalent
- Assembler—Translates an assembly language source program into relocatable code
- Linker—Creates a load module from one or more independently translated modules by resolving the cross-references among the modules
- Relocating Loader—Executes on the host computer and translates the relative addresses into the absolute addresses and produces an execution module
- Run-Time Executive—Provides a variety of operating-system-like services for application programs
- Profiler—Provides a mechanism to monitor the dynamic aspects of an application (e.g., scheduling, Central Processing Unit [CPU] utilization, Input/Output [I/O] channel loading)
- Simulator/emulator—Simulates or emulates the target computer but executes on the host computer and greatly increases testing productivity
- In-circuit Emulator—Emulates, tests, and traces the prototype system operation when connected to the prototype system through the microprocessor socket
- Symbolic Debugger—Allows a programmer to test a module by controlling the program execution on a target computer emulator or the target computer itself using source program symbols or names
- Pretty Printer—Automatically applies standard rules for formatting program source code

- Host-to-Target Exporter—Provides a tool to transmit the execution module from the host to the target when the target machine is different from the host machine

Appendix F provides a more detailed discussion of these tools. A more comprehensive set would include tools covering the following categories: syntax-directed program editing, configuration management, system modeling, relational databases, statistical analysis, Ada program design, software specification and design, software technical documentation, software analysis, and project management. The Naval Air Systems Command Software Engineering Environment (NASEE) tool set contains tools from each of these categories. Appendix A provides information for obtaining these tools. These tools were selected from a number of tool classes and represent the best available in the industry at the time of selection.

4.3 COMMERCIAL Ada DEVELOPMENT TOOLS

Ada has matured significantly over the last few years. Today, more than 416 validated Ada compilers are supplied by 41 vendors. Compilers are available for a wide range of hardware environments ranging from hard real-time applications to business applications. The Ada Joint Program Office (AJPO) updates a list of validated Ada compilers monthly and makes the list available on the AJPO host. Performance on many of these compilers is excellent and compares favorably with that on other compilers.

In addition, more than 300 commercially available tools exist that are backed by more than 200 vendors to support the development of Ada applications across the software life cycle. Tools are available to support both hierarchical and object-oriented design. Design analysis tools are available that use Booch diagrams, Buhr diagrams, Demarco dataflows, entity relationships, flow charts, functional flow diagrams, state transition diagrams, and structure charts. Some of these tools provide automatic code generation from the design diagrams.

Other life-cycle tools support DOD-STD-2167A documentation generation, configuration management, version control, program design, prototyping, reverse engineering, rule checking, debugging, cost estimation, project planning, project management, maintenance analysis, simulation, dynamic memory management, requirements analysis, reuse, dynamic analysis, coverage analysis, frequency analysis, inheritance analysis, metrics analysis, style checker, training, and many others. In addition, tools exist to support source code translation to Ada from Assembly, C, COBOL, DRAGOON, FORTRAN, HOOD Program Design Language (PDL), IRIS, JOVIAL, LISP, and Pascal.

Even Ada artificial intelligence tools are commercially available to support the building of expert systems, knowledge-based systems, natural language systems, and neural networks.

The AJPO maintains an on-line Ada Products and Tools Database that can be used to find out more about these tools.

Ada bindings also are available in commercial products to support the DOD's migration to Open System Environments (OSEs) for the following interface standards: Ada Semantic Interface Specification (ASIS), Generic Package of Elementary Functions (GPEF), Generic Package of Primitive Functions (GPPF), Graphical Kernel System (GKS), Programmers Hierarchical Interactive Graphics System (PHIGS), Portable Operating System Interface for UNIX (POSIX), Structured Query Language (SQL), Transmission Control Protocol/Internet Protocol (TCP/IP), XWindows, MOTIF, Open Look, X.25, X.400, and X.500. Commercial tools are available to support the communication protocols of Ethernet, File Transfer Protocol (FTP), Gateway, Network, and Network File System. (See Section 5.3 and Appendixes G and H for a discussion on Ada Bindings.)

The Ada commercial tool market is projected for continual growth. Dataquest, a company of the Dunn & Bradstreet Corporation, projected that "... software-only revenues for Ada compilers and Ada development environments sold by North American vendors to be $170 million with an annual growth rate of 12–15 percent per year."

4.4 MISSION-CRITICAL COMPUTER RESOURCES ENVIRONMENT

The PSE for a particular project must be selected based on the type of target computer system. The types are Standard Embedded Computer Resources (SECR), commercial computer resources, and Next Generation Computer Resources (NGCR).

4.4.1 Standard Embedded Computer Resources

The Department of the Navy (DON) SECR hardware includes the AN/UYK-43, AN/UYK-44(V), and the AN/AYK-14(V). The Ada Language System/Navy (ALS/N) is the mandated PSE for SECR (see Section 4.6.3 for fuller discussion of ALS/N). Also included with the ALS/N is standard run-time software for execution on SECR hardware.

4.4.2 Commercial Computer Resources

DON policy on the use of nonstandard computers in deployed tactical systems is covered in Tactical Digital Standard (TADSTAND) B. Although a TADSTAND B waiver is required for use of most commercial processors and computers, TADSTAND C still requires the use of Ada (using validated Ada compilers) on all

computer resources not designated as DON standard embedded computers. Currently, more than 385 validated Ada compilers are available for commercial computers through 40 vendors. In addition, 200 vendors provide over 300 specialized products and tools that support software engineering using commercial Ada SEEs. As a result of the Department of Defense (DOD) mandate to use Ada and the more flexible policy regarding hardware waivers, Ada implementation within DON has been greatly accelerated. Currently, many DON projects are using Ada, and many more are committed to using it. (See Appendixes I and J for more information.

One of the first commercial-based computers to be accepted for use without a TADSTAND B waiver is the NAVY Standard Desk Top Computer (DTC II, TAC-3 and follow-ons). Commercial Ada compilers are readily available for DTC II.

4.4.3 Next Generation Computer Resources

The NGCR program, which is described in Section 7.5, will affect the development of the computer resources of the future, replacing and augmenting the current SECR hardware with commercially based computing resources. The new resources will be related through a set of interface standards, and software (i.e., Ada) compatibility will be a necessary part of the family relationship.

Validated Ada compilers will be a necessary precondition for general-purpose processors that use NGCR interfaces. PSEs that support the NGCR philosophy will focus on the emerging commercial standards described in Section 4.6.1.

4.5 AUTOMATED INFORMATION SYSTEMS ENVIRONMENT

The Congressional mandate to use Ada throughout the DON will result in many Automated Information Systems (AIS) applications programmed entirely in Ada. The AIS community uses a wide range of commercial hardware and software. Therefore, the DON policy for establishing the Ada environment for AIS will incorporate the use of Ada into an OSE that will provide the capability to integrate and transport application software across multiple computer systems. The resultant solution is anticipated to accommodate the nonproprietary standards recommended by the National Institute of Standards and Technology (NIST) Application Portability Profile (APP).

4.6 PROJECT SUPPORT ENVIRONMENTS

PSEs are critical to the successful development and maintenance of DON computer systems. Ada PSEs include commercial Ada environments, AdaSAGE, and ALS/N.

4.6.1 Commercial Ada Environments

The commercial Ada PSEs based on validated Ada compilers are steadily increasing in both number and maturity. Commercial Ada PSEs typically contain a set of

system tools that provides file and data management, resource management, and scheduling, and a set of tools that provides the Run-Time Environment (RTE) with loaders, debuggers, and the like. Although the completeness and quality of these Ada PSEs vary, several highly capable Ada PSEs have evolved.

These commercial environments have been exceptionally powerful for the development of Ada software both on host and target environments. A wide range of Computer-Aided Software Engineering (CASE) tools support these commercial Ada environments (see Section 4.6.4.1 for more information). The large number of available Ada PSEs provides Ada users with useful capabilities and alternatives. Once an application is developed on a given Ada PSE, movement to an alternate Ada PSE may require some modification to the associated run-time software.

There is a risk in migrating from one PSE to another. Each contractor's PSE combines a different set of tools and a different overall process to generate the executable Ada code. Moving the entire application from one support environment to another is a difficult task. The industry has not yet evolved the interface standard in these areas.

Even with this risk, the porting of an Ada application from one environment to a different environment is significantly easier than porting non-Ada applications. The use of a single standard applicable to all hardware/operating system environments is an important factor in supporting portability. In addition, use of Ada packages and strong typing provides the means to isolate hardware/system dependencies to significantly facilitate the porting of an Ada application from one environment to another.

4.6.2 AdaSAGE

AdaSAGE is an applications development set of utilities designed to facilitate rapid and professional construction of systems in Ada. Applications may vary from small to large multiprogram systems that use special capabilities. These capabilities include database storage and retrieval (SQL compliant), graphics, communications, formatted windows, on-line help, sorting, editing, and more. AdaSAGE operates on various systems including MS-DOS platforms, UNIX System V, and OS/2. AdaSAGE applications can be run in the stand-alone mode or in a multiuser environment. A developer using the Ada language and the AdaSAGE development system can design a product that is tailored to a specific requirement and offers outstanding performance and flexibility.

AdaSAGE was developed by the Department of Energy at the Idaho National Engineering Laboratory (INEL). In 1987, the U.S. Marine Corps (USMC) was tasked with converting large-scale logistical systems to Ada. The USMC worked with

INEL to include necessary facilities in AdaSAGE. Today, AdaSAGE is part of the USMC Ada PSE. AdaSAGE is primarily suited to support the database-intensive operations found in AIS applications, but it may have some limited use for Mission-Critical Computer Resources (MCCR) applications. Functionality and potential benefits of AdaSAGE include rapid prototyping, programmer usability, and efficiency.

All four Services have reported significant success in developing applications with AdaSAGE. The following projects are complete or nearly complete:

- USMC (Albany, Georgia; Kansas City, Kansas; Quantico, Virginia) Central Design Programming Activities (CDPAs)
  - Marine Corps Integrated Maintenance Management System (MIMMS)
  - Marine Corps Combat Readiness Evaluation System (MCCRES)
  - Supported Activities Supply System (SASSY)
  - Aviation Training and Readiness System (ATRIM)
  - Ground Controller Training System (GTRIMS)
  - Message Edit Processing System (MEPS)

- U.S. Army (Fort Belvoir, Virginia; Fort Monmouth, New Jersey)
  - Unit Level Logistic System (ULLS) ground
  - ULLS air
  - Central Issue Facility (ISM/CIF)
  - In Processing (ISM/InProc)
  - Registration and Access Control System (RACS)

- Navy (Naval Computer and Telecommunications Area Master Station Atlantic [NCTAMS LANT], Norfolk, Virginia; NCTAMS Eastern Pacific [EASTPAC], Pearl Harbor, Hawaii; Naval Regional Data Automation Center [NARDAC], San Francisco, California; and Naval Computer and Telecommunications Station [NCTS], San Diego, California)
  - Casualty Reporting System (CAS REPS)
  - Organization Chain of Command (ORG)

- Air Force—Gunther Air Force Base (Montgomery, Alabama) initiatives

AdaSAGE is viewed by the Services as an extremely valuable tool. Through the Ada Technology Insertion Program (ATIP), AJPO is supporting additional enhancements to this environment, which will make it even more valuable to future users.

4.6.3 Ada Language System/Navy

ALS/N is a software development and run-time environment that is being developed for the current generation of DON standard computers, the AN/UYK-43, AN/UYK-44, and AN/AYK-14. ALS/N also will have the capability to support mixed language applications (i.e., Ada and Compiler Monitor System-2 [CMS-2]).

ALS/N has been validated as Ada/L for the AN/UYK-43 and Ada/M for the AN/UYK-44 and AN/AYK-14. Initially hosted on the VAX series of computers using the VMS operating system, ALS/N will be rehosted to a POSIX-compliant operating system (see Section 7 for further information).

ALS/N consists of two functional parts: the Minimal Ada Programming Support Environment (MAPSE) and the RTE. The MAPSE consists of the compiler and other associated compile-time tools that run on a host computer and produce software products for a target computer. A host computer supports the software development with a general-purpose operating system and file management, resource management, scheduling, and other program development functions.

A target computer is the computer on which the application software (e.g., Command, Control, Communications, and Intelligence [C3I], Anti-Submarine Warfare [ASW], Undersea Warfare [USW], and Electronic Countermeasures [ECM]) will execute. The RTE software provides services needed by the executable application program and supports execution of those programs so as to meet the requirements of performance, reliability, and fault-tolerance on the target computer. The RTE provides the basic and extendible software facilities required to support Ada use in the mission, support, systems, test and maintenance, and trainer software categories. RTE tools include the run-time operating system, executive, librarian, loader, run-time application support, run-time debugger, embedded target debugger, and run-time performance measurement aids. The ALS/N RTE provides run-time support for the AN/UYK-43(V), AN/UYK-44(V) and AN/AYK-14(V) embedded target computers only.

4.6.4 Related Programming Support Environment Activities and Issues

The subsections below provide PSE-related information for CASE tools, the Common Operating Environment (COE), and cohabitation of heterogeneous languages.

4.6.4.1 Computer-Aided Software Engineering

CASE refers to software tools that help automate parts of the software process across the life cycle. These tools are considered to be part of a PSE. CASE tools support the activities associated with a specific part of the system software life cycle, such as requirements specification, coding, and testing. CASE tools also can support project

management activities across the life cycle. Recently, many tools have emerged that support software requirements specification and high-level design. CASE tools can help users organize, document, and generate a specification. Some of the more advanced tools also can execute simulations of the specification and, to a certain degree, generate Ada code or code fragments that fulfill the developed requirements and design. Although many CASE tools will examine the specification and high-level design for consistency and completeness, current CASE tools have widely varying degrees of functionality and maturity. The future of CASE tools is bright and the potential benefits great. Many tools are immature, however, and contractors' claims regarding the capabilities of their tools are often exaggerated. Some CASE tools have difficulty scaling-up to support large software developments (e.g., more than 100,000 Source Lines of Code [SLOC]). Finally, few CASE tools are compatible with each other with regard to method of data transfer or integrated execution.

Many CASE issues are similar to issues surrounding the introduction of Ada. CASE will help impose discipline on the software process, provide better visibility into the software, and encourage the use of modern methods and practices. Three primary benefits of using CASE on a project are improved product documentation, improved project communication, and enforcement of a consistent design and requirements methodology.

Many of the issues surrounding the adoption and use of CASE are organizational, not technical, issues. The organization must have a well-defined software engineering methodology in place before the benefits of CASE tools can be realized. Use of CASE tools often requires a pervasive change in an organization. First, an absolute and strong management support of and commitment to CASE use are needed. Second, selection of high-quality personnel and extensive training are necessary. Third, the resistance to change must be overcome.

In some cases, the initial adoption of CASE requires a large capital investment and, most likely, schedule expansion. This up-front cost in terms of dollars and schedule will be recovered when the lower maintenance costs are realized.

CASE tools are emerging on the market that maintain the specification at a high level and automatically generate Ada code. The intent is to make all changes at the specification level and not at the code level. This technology promises to provide many benefits to software engineering. CASE will surely have an increasing impact on programs developed in Ada and the Ada software development environment.

4.6.4.2 Common Operating Environment

The Navy, Marine Corps, Army, and Air Force have joined in an effort to develop a COE for Command, Control, Communications, Computer, and Intelligence (C4I) systems. The purpose of the COE is to promote interoperability and efficient porting of C4I systems and application software among different hardware and software environments. Today, C4I systems are being designed primarily to support single-Service operations; however, they must perform effectively during joint operations and in support of Joint Force Commanders and their staffs. Systems with similar functions are being developed in parallel and with little coordination among the Services. In joint operations, many C4I functions and sources of information overlap or are substantially the same (e.g., fire support, electronic intelligence processing, and air control), and substantial commonality has already been achieved in air control systems.

Given these conditions, adopting and adhering to a COE for all C4I systems would improve the efficiency of individual Service C4I systems in supporting deployed commanders. Further, a COE would substantially reduce the overall development costs by significantly shortening the time required to develop and field C4I systems. Software applications developed by one Service could be ported to systems of another Service without the cost and delay of translation.

Short-term objectives are as follows:

- To establish a baseline of those constituent elements where commonality already exists
- To define those steps necessary for achieving commonality among all constituent elements

If a COE for the Services appears feasible in the near term, the Navy, Marine Corps, Army, and Air Force collectively will propose expansion of the COE concept and constituent elements to the Joint Chiefs of Staff and DOD agencies for joint implementation.

The mid-term objective is to establish a COE consisting of those elements necessary to describe the environment. It includes, but is not limited to, the following:

- Open system hardware architecture
- Data architecture
- Software documentation
- Local area network and system bus interfaces
- Programming language

- Computer operating system
- Man-machine interface and windowing software
- Graphics interfaces and tool kit
- Map and overlay display, storage, manipulation, and dissemination services
- Real-time tracking database management services
- Encyclopedic database management services
- Security services

The consensus of the proponents is that the COE will permit more economical software development through reuse of Ada software modules and a common look and feel among C4I systems while allowing the Services flexibility to meet their unique requirements. The initial Navy, Marine Corps, Army, and Air Force effort calls for fielded systems using the COE in 1992.

4.6.4.3 Cohabitation of Heterogeneous Languages

DON policy guidance requires the use of Ada in major software upgrades for computer systems. This guidance creates a situation where a choice must be made between redesigning and recoding all of the software in Ada or developing a strategy to support systems with a mixture of Ada and other languages. The other language may be a variant of CMS-2 or a commercial language such as FORTRAN or COBOL.

Completely redesigning and recoding the system in Ada are usually cost prohibitive although necessary to achieve the full benefits of Ada. Simply translating the existing non-Ada code into Ada carries the risk of code expansion and does not take advantage of the many software engineering features of Ada. Cohabitation of Ada code and code in other languages appears to be a necessary option—but it entails risk and requires much corporate planning and commitment.

One approach is to isolate the unique languages by the processor on which they are to run (i.e., use only one language on a given processor). This approach allows interchange of data through message passing or similar methods of data normalization and is a low-risk approach because it minimizes the need to change the old code while eliminating new code constraints. A second approach is to interface existing code to an Ada program using language interface features.

A more problematic technique is to actually mix the languages at the module or program level within a single processor. Among the fundamental problems with this approach is that the run-time operating system must support services for Ada as well as for the other language. Another problem arises because the database and data structures already have been defined by the developers of the non-Ada system. The newly designed and written Ada code must then conform to and be constrained by

that data structure. Unless the Ada code is separable in function and communication paths, the benefits of Ada will be compromised and may not even work. Despite these risks, there have been examples of limited successful cohabitation.

4.6.4.4 Cohabitation of Homogeneous Languages

Another class of problems is encountered when a project tries to mix executable Ada programs from two or more commercial compiler systems on one target computer. In general, such programs are incompatible because of the differences in calling sequences and RTE. One approach to this problem is to take advantage of the portability of Ada programs at the source level. In this way, even if two different compilation systems are used to develop the application software, all source code should be compiled on only one system to create the executable software for the target computer.

4.7 POST-DEPLOYMENT SOFTWARE SUPPORT

Within a computer system's life cycle, the early part of the life cycle is referred to as development, and the later part is called Post-Deployment Software Support (PDSS). Many factors contribute to deciding who actually performs the work involved in PDSS, and these decisions affect the Ada environment. Factors that affect the environment include the following:

- Time required before new requirements are firm
- Delay in availability of details about the requirements change
- Required changes to the compiler and/or validation suite
- Required tool or environment modifications
- Systemwide implications of the Ada modification

For the PDSS portion of the life cycle, work may be performed by the system prime contractor, another contractor, or in-house DON personnel. Each of these options has its advantages and disadvantages.

When the Government has the system prime contractor perform the PDSS, incompatibility problems are avoided between the development and PDSS organizations. The contractor knows and understands the software better than anyone else, hence can maintain it effectively. The major disadvantage is that the Government is locked into perpetual PDSS performed only by the system prime contractor and, therefore, may be unable to control the costs.

Using another contractor or using in-house DON personnel avoids the risk of losing control of costs, but the incompatibility between development and PDSS environments must be addressed. Unless the developing environment and the PDSS environment are the same or are compatible, the transition to PDSS will require

significant effort. Examples exist of transitions to PDSS that have required major code conversions and resulted in high costs and long schedules. One, often costly, alternative is to procure an identical support environment and install it at the PDSS site. Because of the differences among the options, emphasis has been placed on preparing for the life cycle before and during development (e.g., DOD-STD-2167A, 5000 series).

As we enter the new era of Commercial Off-the-Shelf (COTS) and open systems, it will become increasingly important that the DON become a smart buyer capable of handling delivered products. When the development environment and the PDSS environment are different, the potential for problems exists. A key to ameliorating these difficulties and increasing the chance for a successful transition is to require, in addition to delivery of the software itself, a complete quality package including tools and documentation sufficient to understand, regenerate, and maintain the software products. With proper planning, many dollars can be saved during PDSS.

Section 5
Ada Technology Issues

Developing larger, more complex systems requires an awareness of technology issues beyond that required by previous software development efforts. The size and complexity of new systems and upgrades to current systems present engineering and management problems and issues never before encountered. Coordinated efforts in and knowledge of systems engineering, software engineering, Ada products, standards, development, upgrades, and complete life-cycle management as well as software reuse are needed to ensure quality systems.

This section discusses Ada technology issues related to compilers, secondary standards, the transition to Ada, life-cycle documentation, and software reuse. Information on software tools and on the status of the secondary standards is presented in Appendix G of this document.

5.1 Ada TRANSITION

Use of Ada as a standard programming language, plus modern software engineering techniques, will result in more reliable, maintainable, and reusable code. The information below may be useful to an organization that is transitioning to Ada.

5.1.1 Ada Upgrade Opportunities

The future of Department of Defense (DOD) software initiatives and their support base will depend largely on Ada. It is imperative to use Ada for new projects, but it is also wise to search for opportunities to use Ada to upgrade existing projects. Benefits such as ease of maintenance, reusable libraries, variety of tools, and programmer efficiency are a few good reasons to seek Ada upgrade opportunities. Issues that should be considered when transitioning to Ada include the following:

- Compiler maturity
- Run-time efficiency
- Support tools
- Requirements and design impacts
- Training
- Management
- Foreign code interface

5.1.1.1 Compiler Maturity

The maturation of Ada compilers takes time. Immature tools (i.e., those just developed) can have a negative impact on productivity if software developers must

concurrently develop and maintain new applications, new tools, and new hardware. Therefore, the extent to which a compiler has been used in production work should be considered in developing project milestones. When possible, a widely used compiler should be selected.

5.1.1.2 Run-Time Efficiency

Because mission-critical systems must respond to external stimulus quickly and Automated Information Systems (AIS) applications strive to get the most work done on available resources, run-time efficiency should be carefully evaluated. Ways to evaluate run-time efficiency include compiler evaluation, benchmarking, and prototyping.

5.1.1.3 Support Tools

A first objective should be to ensure that a good, basic tool set is used. Many commercially available tools support software developments using Ada. Whether the transition is a complete new start from design and coding or a conversion of code from one compiler to Ada, a good tool set is necessary for an effective transition.

5.1.1.4 Requirements and Design Impacts

Modern software engineering practices are essential to the effective use of Ada. These practices must be integrated at the concept formulation, acquisition, requirements, and design levels. Translating or converting existing code to Ada may produce limited or negative return on investment. The executive (operating system) used must be an Ada run-time operating system or be compatible with the Ada run-time library and Ada Run-Time Environment (RTE). For example, a program's structure depends on the executive selected, and most executives make executive service requests. These service requests (calls) vary with different run-time libraries. Many application and system programs have embedded tasking algorithms that are incompatible with Ada rendezvous and tasking. Interrupt handlers, when used, vary greatly. If the executive is incompatible with the Ada RTE, major changes must be made to the program design or to the executive design. This type of change must be identified in the very early stages of transition and planned for with modification to the requirements and design.

5.1.1.5 Management

Ada offers considerable advantages over other languages used for software development. This technology will continue to evolve to better support software development. Program Managers may have questions about the use of Ada with evolving technology. The ideal way to answer these questions is to train Program Managers to use and develop systems in Ada. However, implementing this alternative is sometimes impossible. In those cases, a Government technical team of experts should be identified to help guide the Program Manager. This team must

understand both maintenance and development issues and, if possible, have a working knowledge of the system requirements and hardware involved.

5.1.2 Mixing Ada With Other Languages

One of the major challenges to the more widespread use of Ada within DOD is the large number of non-Ada systems that are upgraded when necessary. The number of new starts, as compared to upgrades, is very low. Even for new starts, often the strategy is to make the greatest possible use of existing code. This implies that one of the major challenges for the Ada initiative is to develop both viable strategies for inserting Ada into existing systems and a working hybrid model and strategy. With respect to upgrades, there are problems with introducing Ada into these systems, especially when Ada and the original language must reside and execute within the same processor. Some of the issues are data handling, scheduling, and program libraries. If systems require an upgrade, it would be more feasible, from an operational and a technical standpoint, to add processing elements that are "pure" Ada and place greater emphasis on interchanging data. The system upgrade example gains even more importance when considering multiple, heterogeneous processing node expansions to older, non-Ada systems. Architectural issues such as data, program libraries, degraded mode reconfiguration, and networking must be thoroughly investigated. Coprocessing is now progressing from an emerging technology to a practical application consideration.

5.1.3 Reengineering

Reengineering is the modification of one or more elements of a software system while maintaining the system's functional integrity. An example would be conversion of a system from a hierarchical Database Management System (DBMS) to a Relational DBMS (RDBMS). Conversion of a software system from an existing programming language to Ada is considered a form of reengineering. Particularly in the AIS world, a variety of reengineering products are available, and more are under active development.

Organizations and Program Managers responsible for long-term maintenance of DOD software systems should understand the relevance and potential benefits of the reengineering concept. From a management perspective, use of automated tools is the key to the reengineering process. In situations where tools are available, system reengineering can be performed both quickly and economically. Reengineering may be particularly advantageous in situations where large libraries of non-Ada code exist. Because all new systems and any major modifications to existing code must be developed in Ada, organizations with software maintenance responsibilities will be required to maintain expertise in both Ada and the programming languages of their existing systems. In addition, these organizations also face potentially complex and costly integration and cohabitation problems as they attempt to develop and operate

hybrid systems consisting of Ada and non-Ada code. Where reengineering products are available, it may be cost advantageous to convert all existing code to Ada, thereby eliminating the need to maintain long-term programming expertise in other languages. The costs and technical risks of interfacing Ada and non-Ada code also would be eliminated by such a strategy. It should be noted that some reengineering techniques neither provide good readable Ada code nor take advantage of Ada features.

No firm guidance can be given as to whether reengineering is the right option for a particular project or organization. As noted, the commercial market is extremely active in the reengineering products area. Managers need to familiarize themselves with the reengineering marketplace to determine whether reengineering represents a viable and cost-advantageous path for their organizations.

5.1.4 Reverse Engineering

The basic purpose of reverse engineering is to automatically extract the design information for a system from the existing system source code. Currently, reverse engineering is used primarily to generate documentation products to assist in the manual support and modification of systems by using existing source code. The ultimate goal of reverse engineering, however, is to abstract design information from the existing system in a standard design format with an automated tool. A functionally equivalent replacement system could then be automatically generated by the tool selected. Under this scenario, any required change to the system would be accomplished at the design-specification level. Commercial products are emerging on the market to support reverse engineering.

Industry observers generally agree that this type of capability will become available in the near future. The emergence of this type of tool will provide Program Managers with an additional positive option to deal with the hybrid Ada and non-Ada code maintenance problem. Soon, it will be possible to improve the basic design and structure of an existing system, incorporate new requirements, and then regenerate the entire system in Ada. As with the reengineering market, there is extensive commercial activity in the area of reverse engineering products, and Program Managers need to maintain familiarity with available technology.

5.1.5 Porting or Portability

Portability is the extent to which software and data can be transported to different systems. Over the lifetime of an application program, the host development environment frequently changes. The changes occur because of hardware upgrades, modernization, the transition of the support of the life-cycle management function to a different activity, or diminishing or lack of support from the host processor vendor. In these instances, the application and project-developed software tools

would be candidates for porting. Portability must be considered when selecting or developing tools and software. Although Ada applications are, in general, easier to port from one environment to another environment than are applications in other languages, steps can be taken to make porting even easier. The developer must be aware of the portability constraints, isolate the implementation dependencies into a few small packages, and carefully document them so that porting of code will be easier.

5.2 COMPILER SELECTION

A compiler is a software tool or product that receives as input the High Order Language (HOL) or source-language statements developed by designers and/or programmers and translates or compiles these statements into machine-readable, executable code. In their simplest form, compilation systems include only a compiler, linker, loader, library, and fundamental program execution (run-time) structure. More complex compilers include interfaces and support for programmer productivity, testing and configuration management tools, and software development environment structures. These compilers support software development products and methodology as well as project-specific policy.

The compiler selection process should begin with a plan that establishes the project requirements, budget, personnel, and timetable. The criteria for selecting a compiler should be based on the nature of the project; for example, concern about execution time would not be as applicable in a batch-type application as in a tactical program. Based on these criteria, benchmarks, checklists, and interviews should be used as needed to assess different compilers for specific project requirements, and a limited number of candidate compilers should be selected for detailed evaluation.

Evaluating and selecting an Ada compilation system for a project are complex and costly processes. However, evaluation of Ada compilation systems for a particular user application will decrease project risk and reduce total cost and schedule overruns. An Ada compilation system includes the compiler, program library system, linker/loader, run-time system, and debugger. Evaluation and selection apply to the entire software development package, not just the compiler.

To reduce the risk associated with a new language and an immature compiler, the selection process must identify key criteria and test the candidate compilation systems against the criteria. No single test suite or checklist suffices for every project. Several types of benchmarks and test suites can be used to evaluate compiler implementation after specific project requirements are identified. See *Ada Adoption Handbook: Compiler Evaluation and Selection*, Version 1.0. (Weiderman, 1989)

5.2.1 Validation

All Ada compilers must pass formal validation to ensure conformance to American National Standards Institute (ANSI)/Military Standard (MIL-STD)-1815A. DOD Directives 5000.1 and 5000.2 require that all DOD initiatives use validated compilers. Project Managers must select a validated Ada compiler for their software development projects. It is important to verify that compilers are validated against the most recent version of the test suite. (A list of validated compilers can be obtained from the Ada Information Clearinghouse.) The formal validation consists of several hundred tests known as the Ada Compiler Validation Capability (ACVC). A formal validation ensures that an Ada compiler correctly implements the Ada language syntax as defined by the standard. A validation does not, however, assess performance or machine-dependent language features. (See Section 5.2.3 for a discussion of benchmarks.) Ada compiler developers wishing to validate their product may obtain information on the current version of the ACVC test suite from the Ada Joint Program Office (AJPO) or the National Institute of Standards and Technology (NIST).

The timing of compiler procurement should correspond with the start of the project. A validated compiler used at project start is considered validated for the entire life cycle of the designated project. During the project life cycle, it may be desirable to upgrade operating systems, compilers, editors, Computer-Aided Software Engineering (CASE) tools, and the like to the latest version. The Program Manager is responsible for controlling such upgrades wisely because even minor upgrades can have serious cost and schedule repercussions.

5.2.2 Evaluation

Two systems are available for Ada compiler evaluations: the Ada Compiler Evaluation Capability (ACEC) and the Ada Evaluation System (AES). The goal of formal evaluation is to provide vendors, procurers, and users of Ada implementations with comparative compiler performance data. These data enable vendors to improve compiler implementation performance, allow procurers to select implementation and configurations that best meet their project needs, and help users identify the language features that are best to use or to avoid for that particular application.

5.2.2.1 Ada Compiler Evaluation Capability

In 1983, AJPO formed the Evaluation and Validation Team to examine compiler performance issues and provide a capability to assess Ada Programming Support Environments (Ada PSEs) to determine their conformance to applicable standards. As a result, the ACEC was produced. The current ACEC is available from the Data and Analysis Center for Software.

The ACEC strengths include the depth of coverage for language features, documentation, documented structure, code size measurements, timing techniques with statistical model, and cross-system analysis software. The shortcomings are lack of support; limitation of an automated analysis subsystem; weakness in testing compile-time performance; and lack of diagnostics, debuggers, and a library system.

5.2.2.2 Ada Evaluation System

The AES is a test suite designed for the British government to perform testing of an Ada programming environment. Measurements are taken on features such as compile-time and execution-time performance, generated-code quality, compiler-produced error and warning messages, linker and library systems, and debugging capabilities. AES strengths are breadth of coverage, interactive user interface, automatic generation of reports, extensive documentation, macro capability for test generation, checklist for diagnostics, library systems, vendor evaluation, and examples. AES shortcomings are considerable setup time, lack of U.S. support, cost, target run-time performance, coverage, and the subjective nature of checklist.

Currently, plans are to establish an activity to merge the ACEC and AES.

5.2.3 Benchmarks

All compilers are not alike. Benchmarks provide techniques and application examples to compare performance among different Ada compilers or performance among Ada compilers and other language compilers. Careful analysis of the benchmark results helps identify the compiler best suited for the intended project computing environment. To guarantee successful benchmarking, the benchmarks best suited to project requirements must be selected or developed. For some medium and large projects, it may be necessary to develop benchmarks that reflect specific project needs. Program Managers cannot depend solely on publicly available resources; they will have to generate their own benchmarks.

When selecting and developing benchmarks, the Program Manager should ensure that the following apply:

- Benchmarks adequately represent and test the system requirements and the environment selected.
- Benchmarks are part of a planned, total, integrated, supported test suite.
- Benchmarks are maintained throughout the total life cycle.
- Benchmarks and benchmark requirements are suitable for inclusion in a contract.

The Performance Issues Working Group (PIWG) benchmark comprises a suit~ of Ada performance measurement programs that focuses primarily on the execution time of individual features of the Ada language. Many tests in this suite are designed to be machine independent and run without modification. These tests fall into the areas of clock resolution, task creation and rendezvous, dynamic storage allocation, exception handling, array processing, procedural and run-time overhead, composite benchmarks and compilation speed, and capacity test. The strengths of the PIWG benchmark are ease of use, wide use, wide distribution, low run time, and no cost (it is free via the Defense Data Network [DDN]). The weaknesses of the PIWG benchmark are lack of documentation and support.

The Software Engineering Institute (SEI) is currently developing new benchmarks appropriate to Ada application (e.g., Hartstone).

5.3 INTERFACE STANDARDS

It is important that Ada applications are able to access the resources an end user will wish to control. Commonly used resources include databases (e.g., Structured Query Language [SQL], Information Resource Dictionary System [IRDS]), user interfaces (e.g., XWindows, MOTIF, Open Look), Graphics (e.g., Graphical Kernel System [GKS], Programmers Hierarchical Interactive Graphics System [PHIGS]), networked resources (e.g., Government Open Systems Interconnection Profile [GOSIP], X.25, X.400, X.500), hardware (e.g., 1553 data bus), and others. Interfaces to these resources are standardized to promote interoperability, portability, and reuse. For example, the SQL serves as an interface between an application and a database. In theory, use of the SQL standard for databases allows databases to be interoperable from one application to another. Further, an application that uses SQL can be more readily ported from one environment to another. Finally, use of the standard facilitates reuse of the application for new applications.

Both commercial and DOD standards are important to software application development.

The use of commercial standards promotes Open System Environments (OSEs). An OSE encompasses the functionality needed to provide interoperability, portability, and scalability of computerized applications across networks of heterogeneous hardware and software platforms. Data on Federal Information Processing Standards (FIPS) may be obtained by calling NIST. NIST also has produced the Application Pal and other specifications to provide the functionality necessary to accommodate the broad range of Federal information technology requirements. Application Portability Profile (APP) specifications are discussed under operating system services, user interface services, data management services, graphics services, programming services, network services, and data interchange services. The APP is available from

the Superintendent of Documents, U.S. Government Printing Office, Washington D.C. 20402. The APP and FIPS are available from National Technical Information Service (NTIS), Springfield, VA 22161.

DOD standards also may be important to an application, particularly when interoperability is required among the Services or international military organizations. Information on military and DOD standards may be obtained from the Office of the Deputy Assistant Secretary of Defense for Information Systems (C3I), Polices, and Standards.

5.3.1 Bindings

Recently, interface standards were written using an application language as an abstract specification (e.g., Portable Operating System Interface for UNIX [POSIX] basic services was written in C). To become an international standard, an interface standard now must be written in an abstract specification (e.g., VDM SL, Zed). Nevertheless, in order for an application language to use the standard, a language binding to the interface standard is required. Hence, there are Ada, C, FORTRAN, and Pascal bindings to standards such as SQL and GKS. Bindings are required to enable applications to interface with other software products conforming to the standard. Development and standardization of these bindings are ongoing efforts in both national and international organizations such as ANSI and International Standards Organization (ISO).

Usually, an Ada binding to an interface standard is in the form of an Ada package specification. To use the binding in an application, an implementation (normally in the form of an Ada package body) is required. The Ada package is an excellent language feature to support interface standards. Consequently, Ada is an excellent facilitator for OSEs.

There are Ada bindings available today in commercial products to support the DOD's migration to OSEs for the following interface standards: Ada Semantic Interface Specification (ASIS), Generic Package of Elementary Functions (GPEF), Generic Package of Primitive Functions (GPPF), GKS, PHIGS, POSIX, SQL, Transmission Control Protocol/Internet Protocol (TCP/IP), XWindows, MOTIF, Open Look, X.25, X.400, and X.500. Commercial products to support other appropriate NIST APP interface standards are under development or planned (especially CASE Data Interchange Format [CDIF], GOSIP, Initial Graphic Exchange Specification [IGES], and Standard Generalized Markup Language [SGML]). A brief description of the standard and a point of contact for each commercial product that supports an Ada binding is provided in the Ada Information Clearinghouse Report, Available Ada Bindings. Appendix H provides the summary chart from this report, which is updated quarterly.

5.3.2 Secondary Standards

Secondary standards provide an interface to a computational resource. These standards support computation such as mathematical functions, rational numbers, statistical functions, and decimal arithmetic. These computational resources are available commercially to support Ada application development. Use of secondary standards is important to support reuse, interoperability, and portability. For example, transcendental functions may be written with input parameters of either degrees or radians; their return value could be a float type of almost any precision. To maximize reuse potential, a standard set of mathematical functions should be used across an application domain. Secondary standards are emerging. A draft international standard is available for mathematical functions, and other secondary standards will be available soon.

5.3.3 Important Standardization Areas

Important information standards include POSIX, SQL, XWindows, GOSIP, GKS, and PHIGS. Information on their status is provided in Appendix G of this document. Additional information is available in the final report on the JLC software workshop (San Antonio I, Panel VII, 1991).

5.3.3.1 Operating Systems

An operating system is the software or firmware control structure that is closest to the hardware and either provides or supports a specified set of services and functions, formal interfaces, bindings, and resources management to the mission and problem-solving application programs (e.g., MS-DOS, UNIX, VMS). Formal operating systems are available in the most primitive implementation for a single, stand-alone microprocessor as well as for the most complex multiple-node, heterogeneous, time-critical processing systems.

Operating systems reside on host and target processors. Typically, host processors are used to develop software, and target processors run or execute the developed software. Many business and nonmilitary applications and land-based military applications develop and implement (execute) software on the same processor or same family of processors. When the software is developed and executed on the same type of processor, the operating system has an integrated RTE as part of its total operating system. This run-time structure is a specific set of services that supports and executes the selected application software. The application (compiled) program code is "bound" to the operating system at link time, and the entire entity becomes the application.

In some applications, the target processor is not the same processor as the host processor. For these systems, all of the system software services are provided by an

application operating system referred to as the RTE. Ada has been effective for applications with time critical processing requirements on target RTEs.

With regard to standards, validated Ada products conform to MIL-STD-1815A. However, the standard does not resolve run-time issues. The run-time requirements for Ada and languages that use the more conventional operating system products differ in philosophy and structure. Most Ada run-time products are based on proprietary commercial operating system products, and the bindings are usually written in some language other than Ada, such as C or C++.

5.3.3.2 Databases

SQL is an interface standard for use with a standardized language in RDBMSs. In the United States, ANSI technical committee X3H2 standardizes the SQL in two documents: *Database Language—SQL with Integrity Enhancement* (ANSI X3.135-1989) and *Database Language—Embedded SQL* (ANSI X3.168-1989). The next planned revision for the SQL standard is scheduled for 1992. ANSI X3.135-1989 standardizes SQL in a programming-language-independent way; ANSI X3.168-1989 deals with language-specific topics and describes means of embedding SQL statements into Ada programs. The major problem with SQL-compliant database programs is the use of extensions. When database systems allow extensions to SQL, portability is degraded.

At the 7 December 1990 meeting, the International Ada Standards Organization ISOIEC/JTC1/SC22/WG9 passed a motion that endorses the SQL module language interface for Ada as described in Section 8 of ANSI X3.168-1989. This group also passed a motion to support the SQL Ada Module Description Language (SAMeDL) as a binding from Ada to SQL-based databases. (A complete description of SAMeDL can be found in SEI technical report CMU/SEI-90-TR-25.) SAMeDL is designed to facilitate the construction of Ada database applications that conform to the SQL Ada Module Extension (SAME) architecture as described in SEI technical report CMU/SEI-90-TR-25, Section 4. SAME extends the module language defined in the ANSI SQL standard to better fit the needs of Ada. The SAME method involves use of an abstract interface, an abstract module, a concrete interface, and a concrete module. The abstract interface is a set of Ada package specifications containing the type and procedure declarations to be used by the Ada application program. The abstract module is a set of bodies for the abstract interface. These bodies are responsible for invoking the routines of the concrete interface and converting between the Ada and the lower level data and error representations. The concrete interface is a set of Ada specifications that defines the SQL procedures needed by the abstract module. The concrete module is a set of SQL procedures that implements the concrete interface.

5.3.3.3 Graphics

The Graphics System Interface Standard (GSIS) is one of the sets of standards that is essential to the timely and cost-effective acquisition of the majority of DON's next-generation, mission-critical computing systems. The GSIS will help DON to efficiently provide graphics systems that address a wide range of functionality for various levels of military graphics applications.

An interactive graphics system is defined as a set of hardware and software that together provides the following services and physical devices:

- Accepts human input (e.g., via a mouse, keyboard, buttons, touch screen) and presents it to the application program via an accepted protocol (procedural interface) and in an accepted representation

- Presents data and information (e.g., display formats and images on cathode ray tubes) to its human users via a protocol adhered to by the application program and the graphics system as the provider of graphics services

Some candidate graphics system interfaces being considered include the following:

- Computer Graphics Interface (CGI)
- Graphic Kernel System (GKS)
- Programmers Hierarchical Interactive Graphics System (PHIGS)

CGI, formerly the Virtual Device Interface (VDI), is a device-level interface. The CGI standard establishes a universal interface between higher level graphics software standards, such as GKS and PHIGS, and lower level device drivers. Presently, CGI is a two-dimensional standard although work is being done to produce a three-dimensional version. CGI is largely a European effort and has been accepted by the ISO.

GKS is a portable, device-independent graphics interface package for two-dimensional graphics output and interactive input. It consists of a library of routines that an application programmer can incorporate into a graphics program to produce and control pictures. GKS obtains its device independence through its definition of a virtual device. The virtual device specifications are translated into physical device specifications when the application program is executed. Currently, efforts are under way to add a three-dimensional extension to GKS. Both ISO and ANSI have accepted GKS.

The PHIGS standard defines a portable, hardware-independent application interface for two- and three-dimensional graphics output and interactive input. PHIGS, an

evolutionary product of GKS, has an extensive set of tools that provides an efficient and useful structure and the capability to edit and debug programs after they have been written. PHIGS is both an ANSI and an ISO standard.

5.3.3.4 Windowing Environment

The XWindow system is a hardware-independent and operating-system-independent graphics standard designed to operate over a network or within a stand-alone machine. Developed at the Massachusetts Institute of Technology in 1984, it has become an industry standard employed by such companies as American Telephone & Telegraph (AT&T), Digital Equipment Corporation, Hewlett-Packard, International Business Machines (IBM), SUN Microsystems, and others.

The libraries (Xlib and XT) are rudimentary in scope and provide a basic communication protocol for XWindows, which are in turn used by higher level tool kits (e.g., MOTIF, Open Look) to facilitate the writing of user interfaces. (More information on these libraries can be obtained from the Software Technology for Adaptable, Reliable Systems [STARS] office.) Additional capabilities may be added by layering a tool kit over Xlib. Tool kits generally have routines for building menus, push buttons, slider controls, and the like. Several different tool kits that conform to either the MOTIF or the Open Look graphical user interface standard are available. Ada bindings to the Xlib and XT XWindow interface are available free on the STARS repository identified in Appendix A.

5.4 SOFTWARE REUSE

Software reuse is the process of using preexisting software and documentation to implement new software systems. The general belief is that software reuse will become more widespread as technical and ownership issues are resolved. This subsection surveys estimates of the economic benefits of software reuse and summarizes some reuse techniques and issues that apply to Ada software development.

5.4.1 Economic Benefits of Software Reuse

Attempts have been made to quantify the economic benefits of software reuse for software development. Potential estimates of savings range from 50% to greater than 90%. Most experts on this topic agree that because software costs are not concentrated in any particular development phase, reuse techniques should be applied across all phases of software development (i.e., requirements definition, design, code production, and test) to achieve these potential savings. SEI recently has published economic models that estimate cost savings due to software reuse under a variety of conditions. The SEI models provide a qualitative analysis of conditions needed to make software reuse economically beneficial.

5.4.2 Software Reuse Techniques Applicable to Ada

This subsection samples current techniques in software reuse that can be applied to Ada. These techniques are not mutually exclusive; successful projects have adopted concepts and methods from more than one of the following reuse techniques:

- Classification techniques
- Design techniques
- Programming-in-the-large
- Commercial Off-The-Shelf (COTS) software

5.4.2.1 Classification Techniques

Classification techniques attempt to describe how a software repository is organized so that components may be easily identified and retrieved. Domain analysis, a method by which an application domain is decomposed into component processes, is one such technique. The resulting collection of connected processes may serve as a standard for organizing a reuse library of that domain. Additional information on domain analysis is available in "An Object-Oriented Approach to Domain Analysis" (Shlaer and Miller, 1989). This technique and several similar techniques describe a reuse repository where identifying information is stored in n-tuplets and accessed by a query system. Additional information on this technique can be found in "Classifying Software for Reusability" (Prieto-Diaz and Freeman, 1987).

5.4.2.2 Design Techniques

Information hiding is a design technique that allows software costs to be significantly reduced by keeping software changes as localized as possible. This design technique can have a positive impact on software reuse because well-designed Ada packages containing few input parameters (hence, less need to know the environment external to the component) are more likely to be reusable. A common example would be the abstraction or hiding of device-dependent logic from other portions of the program so that the other portions may be reused easily with different devices.

5.4.2.3 Programming-in-the-Large

Research in the area of programming-in-the-large attempts to address the building of large, complex programs by defining a formal grammar construct or language for specifying the assembly of components. An Ada-like specification language should be used. Other languages that provide such specification in terms of intercomponent interfaces are called module interconnection languages. A surveyed list of these languages is provided in "Module Interconnect Languages" by Ruben Prieto-Diaz and James M. Neighbors.

5.4.2.4 Commercial Off-The-Shelf Software

Reusing COTS software is another software reuse technique that can be applied to Ada. Ada software can be reused at levels other than the component level; that is, entire programs and collections of programs may be reused. Recently, some Government agencies have adopted this practice and are distributing software for reuse by other programs. Reuse of entire programs simplifies many issues: a repository is not required because the entire software package is reused, configuration control is streamlined because all users receive the same software version, and production planning can be simplified through a periodic release cycle. This form of reuse, however, may be more restrictive because users may not be permitted to modify the reused software. Lack of detailed visibility and control of such COTS products can severely hamper accomplishment of software safety and mission security requirements.

5.4.3 Management Issues

Among the incentives for increasing the use of software reuse are increased productivity (less code development required) and increased quality (use of previously tested code). Inhibitors of software reuse include lack of trust in a code developed elsewhere; the desire to use the latest innovative language, tool, and approach; and lack of knowledge about or difficulty in obtaining information on available tools, software, or repositories that can be reused.

The issue of data rights is sometimes difficult to address. A piece of software written by an employee of a company on company time generally belongs to the company. In addition, if a piece of software is contracted out, the contract must specify who owns the software upon delivery. The issue becomes clouded when a contractor or employee borrows software from some other source for use on a contract. In some instances, the original source is not clear. Generally, it is a good practice for any purchased software contract to specify who owns the software. Another data rights issue that should be considered is the responsibility for software that does not work correctly. This is an issue that the Government faces with Government-Furnished Equipment (GFE). When the software is GFE and it does not work, the Government may be responsible for any milestone slips or additional cost resulting from the software problem.

To fully reuse software, a one-to-one correspondence is needed between specifications, requirements, design, code, and test procedures. With this correspondence, not only can code be selected from a repository, but also its associated documentation can be reused. This correspondence also is helpful to the user. It defines what the software does and helps determine whether its design is compatible with its proposed use.

In developing reusable code, special attention must be paid to the parameters and structure of the software units to be reused so as to isolate specific project hardware and dependencies. The software must be developed with reuse in mind. Ada packages help in this area, but the code must be designed carefully so that the reusable unit is as generally applicable as possible.

As the units of reusable code increase and become generic, more automated tools are required to keep track of the reusable pieces. In some cases, the automated tools to build reusable systems are equally or more complicated than the systems they build.

Evidence that widespread software reuse is achievable without hampering significant technology advances is provided by the Japanese software factory experience in which software reuse is seen as essential for economic success. Clearly, the features of Ada and current software engineering technology make it possible to exploit software reuse and achieve significant gains in software productivity and quality.

5.4.4 Reuse Repositories

The Reusable Ada Products for Information Systems Development (RAPID) Center minimizes, identifies, and isolates implementation-dependent characteristics of Ada software, thus creating and promoting reusable code. The certification process includes defining the types of components for reuse, processing the components through RAPID's developmental phases, and assigning a RAPID certification level to the components. The RAPID Center supports the RAPID Center Library (RCL), which is an automated catalog and retrieval system that allows a user to identify and extract Reusable Ada Software Components (RSCs) that meet specific functional requirements. More information about RAPID can be obtained from the U.S. Army Information Systems Software Development Center in Washington, D.C. See Appendix A for information on contacting RAPID.

5.5 LIFE-CYCLE DOCUMENTATION

The requirements for life-cycle documentation are specified in DOD-STD-2167A for Mission-Critical Computer Resources (MCCR) systems and in DOD-STD-7935 for AIS systems.

5.5.1 DOD-STD-2167A

DOD-STD-2167A is a tri-Service standard that contains the requirements for developing mission-critical software. It also provides for maintainability and transition from the development to the maintenance phases. Government Program Managers must cite DOD-STD-2167A in contracts, specifically in the SOW standards and the associated Data Item Descriptions (DIDs) on the Contract Data Requirements Lists (CDRLs). The standard is meant to be tailored for each

individual program. Tailoring guidance may be found in Military Handbook (MIL-HDBK)-287, "A tailoring guide for DOD-STD-2167A." This standard was issued on 29 February 1988 and is scheduled for revision in 1993.

A Personal Computer (PC)-based tool, Tailor/2167A, is available from LOGICON to help Program Managers tailor DOD-STD-2167A. The Joint Logistics Commanders' Computer Resource Management Panel obtained a license from LOGICON to distribute the tool to DOD and Service Program Managers. A PC-based tool, TAILOR/DIDs-2167A is also available to Government managers. This tool helps managers tailor the 2167A DIDs. DOD and Service managers can obtain a free copy of these tools from their respective Service representative:

- U.S. Navy—Space and Naval Warfare Systems Command (SPAWAR)-3212, Washington, D.C. 20363-5100

- U.S. Marine Corps—MCTSSA/AQA, Camp Pendleton, CA 92055-5080

5.5.2 DOD-STD-7935A

DOD-STD-7935A was published to provide for AIS life-cycle maintenance. This standard addresses life-cycle maintenance from beginning to end and covers project justification, concept, design, implementation, and documentation. Several Navy instructions expand and amplify this instruction. Two of the more frequently used instructions are Secretary of the Navy Instruction (SECNAVINST) 5233.1B and Navy Data Automation Command Publications (NAVDAC PUBS) 24.0, 24.1, and 24.2.

Section 6
Lessons Learned

This section summarizes the collective set of lessons learned on more than a dozen Department of Defense Ada projects, including the following:

- Advanced Field Artillery Tactical Data System
- AN/BSY-2 Submarine Control System
- Ada Language System/Navy Full-Scale Development Program
- Avionics Project for an airborne Command, Control, and Intelligence application
- PEO-SSAS, PMS-414, SEA LANCE
- Navy World Wide Military Command and Control System Site-Unique Software Project
- Event-Driven Language/COBOL-to-Ada Conversion Program
- Shipboard Gridlock System with Auto-Correlation
- Combat Control System MK2
- P-3C Update IV Ada Development
- Standard Financial System Redesign
- Reconfigurable Mission Computer Project
- Intelligent Missile Project

Appendix I provides a full description of these projects and the lessons learned on each. This appendix also provides a matrix showing the lessons learned by specific category and project. As review of this appendix shows, most of the problems encountered were management-related, not Ada-related, problems. In addition, some problems recur across all of these projects, such as:

- Lack of training and/or experience
- Failure to take a risk engineering approach
- Improperly specified contract requirements for software-related items and processes
- Inadequate estimates of resources and/or facilities needed
- Immaturity of Ada development tools and environments
- Insufficient [lack of] incremental testing

The subsections below highlight a few of the lessons common to several of the projects in the areas of standards and policy, project management, development process, corporate knowledge and software development experience, training, resources and facilities, support environment tools, reuse, and project costs.

Before undertaking any software-intensive system development, the reader should review the matrix in Appendix I, and the Project Manager should study the detailed project descriptions in this appendix so as to benefit from the lessons learned.

6.1 STANDARDS AND POLICY

Review of the project lessons shows the necessity for establishing a policy to ensure that planning and monitoring of software development occur early in the development process. The lessons also highlight the importance of incorporating the critical elements of the Military Standards into the acquisition package (e.g., the Request for Proposals). In addition, the lessons suggest that policies be established to require incremental build and testing, use of metrics from the beginning of the project, and development of a common style guide for use across development teams.

6.2 PROJECT MANAGEMENT

Nearly every lesson learned in these projects related to project management, as the matrix in Figure I-1 shows. Among the recurring lessons in this area is the need for up-front planning and close monitoring of the project. Also evident, and related to up-front planning, is the necessity for ensuring that adequate facilities and resources are available.

6.3 DEVELOPMENT PROCESS

The most significant lesson learned across projects was the importance of applying sound system engineering and software development practices and principles at every stage of the project. Of particular importance are strict configuration management

and quality assurance. On several projects, use of a consistent methodology and adoption of a risk engineering approach were mentioned as key to the development process.

6.4 CORPORATE KNOWLEDGE AND SOFTWARE DEVELOPMENT EXPERIENCE

In this area, the lessons learned emphasize that both Government and contractor developers must understand the project requirements and adhere to them. Corporate knowledge and software development experience also are needed to establish schedules, determine at what point full-blown coding should begin, and identify the resources available to meet system requirements.

6.5 TRAINING

On several of the projects, the need for Ada-specific training was noted because most of the experienced personnel have little or no experience with Ada and modern software engineering practices. This need for training applies to both technical and management personnel. Project experience suggests that hands-on training should be conducted as close as possible to development or during development.

6.6 RESOURCES AND FACILITIES

As mentioned above, review of the project lessons indicates that the initial estimates of the resources and facilities needed often were inadequate. Project experiences showed the necessity for having contractors identify the tools to be used in development and/or conducting a system analysis to ensure that the adequate resources will be available to meet requirements.

6.7 TOOLS

For large, geographically dispersed projects, the lessons learned show that common support tools should be required. Use of common tools allows problems to be identified quickly, workarounds made only once, and results entered into a shared electronic reporting system. In addition, before committing to large projects, the methods and tools should be exercised; the team must be well trained in the use of the supplied tools; and the tools must work as advertised. Project experience also indicates that use of automated tools should be mandatory for large software undertakings and that development tools are essential.

6.8 REUSE

The project lessons show that Ada facilitates reuse and that planning for and designing in reuse yields long-term benefits. It was noted that development and maintenance time can be reduced significantly by capitalizing on reuse. However,

project experience suggests that large-scale software component reuse will depend on achieving more technological progress.

6.9 PROJECT COSTS

On several projects, the effect of inadequate initial estimates of the needed resources had cost implications. In some cases, additional funding was needed for support hardware and facilities and for training as well as to accommodate schedule delays.

Section 7
Future Directions

The ability of the Department of the Navy (DON) to acquire and maintain effective software-based systems is closely linked to its desire and commitment to changing the way it has conducted business in the past.

DON has already realized the potential of Ada and has made significant progress in the use of Ada for software development in its mission-critical community. The future of Ada within DON is promising, as evidenced by the interim DON policy on Ada (dated 24 June 1991), the Next Generation Computer Resource (NGCR) Program, the Ada Language System/Navy (ALS/N) Project, Tactical Digital Standards (TADSTANDs), and several Ada projects that are planned, completed, or under development. The available supply of Ada-based software management, engineering tools and support environments is increasing as a result of continued commercial investments. The supply will continue to grow to better meet most specific mission domain needs for Mission-Critical Computer Resources (MCCR) Systems and Automated Information Systems (AIS).

Open system interface standards will be used when they have been further defined and validated through the NGCR Program. Research initiatives, such as those being conducted by the Software Engineering Institute (SEI), the Defense Advanced Research Projects Agency (DARPA), and Software Technology for Adaptable, Reliable Systems (STARS), are advancing software engineering technologies. These initiatives will provide new opportunities for improvement in the DON software development process.

The future of Ada in the DON is assured and supported by the following Department of Defense (DOD) initiatives:

- Ada Technology Insertion Program (ATIP)
- Ada 9X
- Ada Reuse
- Ada Language System/Navy (ALS/N)
- Next Generation Computer Resources (NGCR)
- Common Ada PSE Interface Set-A (CAIS-A)
- Portable Common Interface Set (PCIS)
- Software Technology for Adaptable, Reliable Systems (STARS)
- Corporate Information Management (CIM)
- Software Engineering Institute (SEI)

- Several plans, including the Software Action Plan (SWAP), Software Technology Plan (SWTP), Computer Resources Strategy, and the DON Training Guide

7.1 Ada TECHNOLOGY INSERTION PROGRAM

The Ada Joint Program Office (AJPO) sponsors the ATIP to provide risk reduction for the insertion of Ada technology into DOD systems. Originally, ATIP was intended to accelerate Ada usage into programs currently developing systems through direct cost-sharing assistance. Today, ATIP is focused on accelerating Ada usage by addressing education, binding, and technology issues to benefit the entire Ada community.

ATIP projects were solicited through the Ada Executive Officials on 11 January 1991. Many excellent proposals were submitted. The DON alone submitted 26 ATIP proposals by the requested deadline. Although only 5 of the 14 selected proposals were submitted through the Navy, each of the selected proposals will benefit the Navy in the long run.

Selected proposals will provide benefits to a wide range of DOD applications. Their cost can be effectively leveraged to reduce cost to other software engineering activities. An excellent example of ATIP leveraging was provided by the Tool for Multi-Task Input/Output (I/O) Real-Time Message Formatter project at the Naval Ocean Systems Center (NOSC). This project, sponsored by FY89 ATIP funding, was intended to reduce risk for advanced Ada usage as part of the North Atlantic Treaty Organization (NATO) Interoperable Submarine Broadcast System (NISBS). Combined with software from the Advanced Message Processing System (AMPS), the project provided a basis for reusing software effectively for the Submarine Message Buffer (SMB) Program. The $130 thousand spent for this project resulted in projected savings of approximately $2 million for the SMB Program. It also resulted in a significantly reduced schedule to field a high-quality product to the fleet.

The ATIP projects selected for FY91 fell into three categories: education, bindings, and technology. The education project will result in introducing Ada and software engineering curricula into our colleges and universities. The bindings projects will enable DOD programs to take advantage of standard interfaces for their software developments. Ada Binding projects supported include Government Open Systems Interconnection Profile (GOSIP), Management Information System Mathematical Bindings, Military Standard (MIL-STD)-1553, Portable Operating System Interface for UNIX (POSIX), Structured Query Language (SQL), and XWindows. The technology projects will advance the state of the technology in the areas of reuse, prototyping, security, and Ada software engineering environments. Appendix J

provides a breakdown of these ATIP projects by category and a brief description of as well as a point of contact for each project.

The ATIP projects were selected to reduce cost to future DOD and DON software engineering activities. Points of contact for these projects can provide you with valuable information today that may benefit the planning of your program.

7.2 Ada 9X

The tenets of both the American National Standards Institute (ANSI) and the International Standards Organization (ISO) require that each standard be periodically revisited. Ada 9X is the effort to perform this function for the Ada programming language.

7.2.1 Background

The Ada language, ANSI/MIL-STD-1815A, was published in 1983. Starting in 1984, the number of available Ada development facilities began to increase. As Ada was rigorously used in several projects, a series of omissions, limitations, and minor errors were identified. In January 1988, the AJPO asked the Ada Board for a recommendation on how to resolve this situation.

In September 1988, the Ada Board delivered its report, which recommended that the language be revised. A project office to accomplish this task was established under the direction of Christine M. Anderson at Eglin Air Force Base, Florida. The goal of the project is to revise Ada 83 and effect a smooth transition from Ada 83 to Ada 9X (ANSI/MIL-STD-1815B). During the project, a public survey was conducted to solicit revision requests, and more than 750 revision requests were received. Several international workshops were convened to review and prioritize those inputs.

Changes will be constrained by the overall objective of minimizing negative impact and maximizing positive impact on the Ada community. The effect on managers, programmers, vendors, educators, authors, and various application domains will be considered during the transition.

The revision will include only those changes that improve the usability of the language while minimizing the disruptive effects of changing the standard. The revision process will continue and will include various forms of public scrutiny such as conferences, electronic mail, and draft documentation. ANSI and ISO approval of the revisions is expected in 1993.

The proposed revision requirements, which were completed in December 1990, are categorized into the following areas:

- General requirements—Collection of small defects in the language with the structure and format of the standard retained
- Real-time requirements—Precise control over when an action takes place
- Systems programming Requirements—Machine operations, data interoperability, interrupt entry binding, and operations on pointers
- Safety-critical trusted requirements—Ability to analyze generated code for certification and to provide correspondence between the source and the generated code
- Support of programming paradigms—Subprogram manipulation, data storage control, recompilation, object-oriented programming support, and generic modifications
- Parallel/distributed processing (capability currently does not exist)—Distribution of single programs, distribution of an Ada system, remote communications, and configuration control
- Information systems—Currency quantity handling, character set compatibility, interface to Database Management Systems (DBMSs), and common data structures
- Scientific and mathematical applications—Location of point and data storage
- International user requirements—Topics such as international character sets

7.2.2 Ada 9X Revision Activities

The requirement will be mapped into language solutions, and the wording in the standard will be revised. Three major enhancements are planned: support for object-oriented programming, programming-in-the-large, and a light weight synchronization mechanism.

7.2.3 Ada 9X Transition Activities

Transition activities involve management, programmers, vendors, and Ada Compiler Validation Capability (ACVC) test suite revision and policy.

7.2.3.1 Managers

To help managers transition to Ada 9X, two Ada 9X managers workshops will be conducted before ANSI/MIL approval: one for mid-level management and one for executive-level management. Transition issues and strategies will be discussed.

A short (approximately 15–20 minute) videotape that discusses the language in terms of corporate benefits and policy issues will be developed for managers.

A concise guide to practical steps for transitioning to Ada 9X from both non-Ada and Ada 83-oriented organizations will be developed. The guide will be similar to the Ada Adoption Handbook for Ada 83 developed by SEI. It will include a discussion of the benefits of using Ada 9X from a manager's perspective, tips on tool selection, and a summary of policy.

7.2.3.2 Programmers

To help Ada 83 programmers transition to Ada 9X, an "Ada 9X Programmers Guide" will be developed. This guide will highlight, chapter by chapter, the changes between Ada 9X and Ada 83 and discuss programming strategies that use new features. Any incompatibilities between Ada 9X and Ada 83 will also be noted, and straightforward modifications to Ada 83 code will be provided to transition to equivalent legal Ada 9X code. Suggested Ada 83 coding practices to facilitate the transition to Ada 9X also will be discussed for those programmers who are continuing to use Ada 83 on existing projects. A 1-hour videotape also will be developed that will highlight the changes to the language and will feature opportunities for use as well as programming examples.

7.2.3.3 Vendors

During the Ada 9X revision process, several vendor workshops will be conducted. The purpose of these workshops will be to allow vendors to closely track the revision and to provide feedback to Ada 9X teams regarding implementability. An electronic vendor bulletin board has been established to allow vendors to interact directly with Ada 9X Project team members. Open and direct dialogue is essential for a timely and effective transition.

7.2.3.4 ACVC Test Suite Revision

AJPO has frozen the Ada 83 test suite (ACVC 1.11). For information only, a baseline for the Ada 9X ACVC Test Suite, 9XBasic, will be released approximately 12 months before ANSI approval. It will eliminate Ada 83/Ada 9X incompatibilities and focus on usage-oriented tasks rather than remote fringes of the language.

The first Ada 9X ACVC release will be designated ACVC 2.0 and will cover part of the new Ada 9X features. ACVC 2.1, which will incorporate the remaining tests for Ada 9X, will be released approximately 9 months after ANSI approval. The Ada 9X test suite will focus on usage. Figure 7-1 provides the planned release schedule.

Figure 7-1. ACVC Schedule

Suite	Objectives Available	Tests Available	Start	End	Certificate Expiration
2.0	2 MAC	3 MA9X	3 MA9X	27 MA9X	36 MA9X
2.1	3 MA9X	9 MA9X	27 MA9X	63 MA9X	75 MA9X

MAC = Months after release of Ada 9X ANSI canvass for voting
MA9X = Months after ANSI approval of Ada 9X

7.3 Ada REUSE

Several efforts currently underway in DOD are addressing software reuse (e.g., STARS, CIM, SWAP), all of which recognize that software reuse has the potential to yield substantial improvements in the quality and reliability of DOD software systems at a reduced cost. The main objective of all of these efforts is to create an environment in which Program Managers can take advantage of reusing already developed software components as an alternative to developing new code. The reuse concept, however, raises several issues that must be addressed and resolved, including the following:

- Policy and regulations that inhibit software reuse
- Incentives to developers, Program Managers, and contractors to reuse existing software
- DOD infrastructure to facilitate widespread software reuse
- Cultural change in the areas of software development, acquisition, and support to accept and promote reuse
- Data rights provision
- Work force education in areas of reuse technology
- Technology to support confident composition of software components

7.4 Ada LANGUAGE SYSTEM/NAVY

In the next few years, the ALS/N software tool set will be modified to provide continued "software upgrade" capabilities for the more than 3,000 processors and/or systems deployed throughout DON.

A post-deployment enhancement program is underway that will continue through 1995. This program will target an Enhanced Processor (EP) AN/UYK-44, the Very High-Speed Integrated Circuit (VHSIC) AN/AYK-14 and the High-Performance Processor (HPP) AN/UYK-43. Also underway, in anticipation of the transition to the NGCR program and POSIX, will be a UNIX rehost activity.

7.5 NEXT GENERATION COMPUTER RESOURCES

The NGCR program has been established to eliminate shortcomings associated with the current DON Standard Embedded Computer Resources (SECR) Program. These shortcomings include outdated technology, fixed architectures, limited processing capability, poor size and performance ratios, and lack of cost share benefits to develop software support environments. The NGCR program's goal is to provide computer resource standards that meet DON mission-critical requirements in the late 1990s and beyond. These standards are not single computer or software systems; they are hardware and software interface and protocol standards that can be used to certify any number of computers and software systems. The NGCR program will establish the computing system architecture, functional interface standards, and acquisition methods that will govern the acquisition of a family of computing resources to cover most of DON needs into the 21st century.

The program focuses on the identification of nonproprietary, widely used standard bus and external interfaces that can provide the capability to build larger processing suites by the aggregation of sets of industry-produced, DON-approved hardware and software components that can be integrated for enhanced operation. With standard and well-defined interfaces, this building block approach will allow component technology to evolve without the need for revolutionary changes in system architecture and packaging.

The NGCR effort will attempt to augment the Open Systems Architecture (OSA), approach to computer acquisition. For OSAs, the internal and external hardware and software interfaces, services, and protocols are well specified; they have undergone public review and have been published and widely accepted as standards by organizations such as the Institute of Electrical and Electronics Engineers (IEEE), ANSI, and ISO; and they are implemented in vendor products. This approach will allow DON to take advantage of existing and future industrial competition for Navy computers on a continuing basis, which will dramatically improve the technical and operational performance of DON computer-based systems.

In August 1988, the NGCR effort identified the following standard areas for investigation:

- Backplane (BP), 1992*
- High-Performance Backplane (HPBP), 1997
- High-speed Data Transfer Network, 1994
- Local Area Network (LAN)—Survivable Adaptable Fiber-optic Embedded Network (SAFENET I) (LAN), 1990*
- LAN—SAFENET II, 1992*
- High-speed LAN, 1998
- Operating System (OS), 1996, 1998 (MLS)*
- Database Management System (DBMS), 1998
- Project Support Environment (PSE), 1998
- Graphics Language Interface

Of the 10 areas, only the BP, the LANs, and the operating system (i.e., items marked with an asterisk) will be actually prototyped and conformance tested. A formal conformance testing capability will be established within DON and will be available for acquisition managers by 1996.

7.5.1 Project Support Environment Standard Working Group

Consistent with the objectives of the overall NGCR program, the goal of the Project Support Environment Standard Working Group (PSESWG) is to establish DON standards for PSE interfaces that will enhance the DON's ability to acquire PSEs quickly and cost-effectively. Also consistent with the NGCR program guidance, the PSESWG will be a joint team composed of members from DON, other Government institutions, industry, and academia.

PSE interfaces selected for standardization will include data interchange formats and interfaces to the user, the DBMS, life-cycle process management, and the network. Because the objective is to standardize interfaces based on industry standards, the PSESWG work will not select particular tools or products. The group will pursue the

adoption of interfaces with Ada language bindings as well as those for other languages, such as C.

The PSESWG will coordinate with several other important groups in the environments community, including SEI, STARS, and the National Institute of Standards and Technology (NIST), in addition to the other military Services. This coordination is expected to yield the maximum benefit and reduce duplication of effort. Because of the wide range of interfaces to be considered for standardization by PSESWG, the standards are expected to emerge incrementally, perhaps as early as 1994, and work will continue until approximately 1998.

7.5.2 POSIX Standard Working Group

The Navy's NGCR program and the National Aeronautics and Space Administration (NASA) have selected the POSIX standard as the nonproprietary operating system interface standard. The IEEE P1003 POSIX Standard Working Group is developing standard Ada bindings and language-dependent bindings for POSIX-compliant products. The NGCR Operating Systems Standards Working Group (OSSWG) is participating in the POSIX standardization effort in order to support future Navy requirements.

7.6 COMMON Ada PROGRAMMING SUPPORT ENVIRONMENT INTERFACE SET

The CAIS, MIL-STD-1838A, was developed to serve as the framework for an Integrated Project Support Environment (IPSE). It is based on an Entity Relationship Attribute (ERA) model that provides tool-to-tool integration services at a level higher than an operating system.

IPSEs will be critical to the acquisition of future software systems. They provide an environment in which tools can work together effectively to support the development of application software for the total software development life cycle. Their strengths include interoperability and portability of tools, databases, and personnel from one environment to another, which result in higher quality application software that is produced on schedule and within budget.

CAIS-A is the interface technology behind a NATO program that currently is developing a usable IPSE. The capability to have interoperability and portability of tools and databases is very important to NATO because software applications are frequently developed as multicountry cooperative efforts that involve many contractors and subcontractors. This program, managed by a Special Working Group (SWG) on Ada PSE, is providing a suite of tools for VAX/VMS CAIS-A and SUN/UNIX CAIS-A hosted environments. The tools include the following:

- Ada compiler (Germany)
- Linker (The Netherlands)
- Symbolic debugger (Norway)
- Syntax directed editor (Canada)
- Command language interpreter (Canada)
- Print tool (Canada)
- Test tools (The Netherlands)
- Requirements analyzer (Italy)
- Configuration management tool (Spain)
- Mathematics package (United States)

Cross-compiler/debug capabilities to the MC 68020 also have been produced. In addition, the STARS program has developed several CAIS-A hosted tools including a graphical browser and a reuse library tool.

In 1992, this interface technology will be demonstrated using two different weapons systems scenarios. An evaluation of the technology will be completed during 1993. CAIS-A interface technology is expected to benefit both MCCR and AIS application development. The Army's Strategic Defense Initiative is positioning itself to take advantage of CAIS-A hosted environments.

Useful CAIS-A references include the "Common Ada Programming Support Environment (Ada PSE) Interface Set (CAIS) (Revision A)" (Ada Joint Program Office, 1989) and "Introduction to CAIS (MIL-STD-1838A)" (Hitchon et al., 1989).

7.7 PORTABLE COMMON INTERFACE SET

The PCIS Program is a NATO effort sponsored by the Special Working Group on Ada Programming Support Environments (SWG on APSE). This program will define framework-level services for an Integrated Software Engineering Environment (ISEE). This framework will allow tools to be truly integrated so that they work together effectively to support the development of a software application. These framework services will be based on requirements identified in the International Requirements and Design Criteria (IRAC) document and the European Computer Manufacturing Association/National Institute of Science and Technology (ECMA/NIST) Reference Model. The services will include object-management, process-management, communication, and user-interface services. The PCIS framework will be based on the ECMA Portable Common Tool Environment (PCTE) to which a standard Ada binding exists as ECMA Standard 162. The PCIS Program will evolve the framework services in the French Enterprise II. Plans are to complete the PCIS framework definition by the end of 1993. A prototype and APSE demonstrator are planned for early 1994.

7.8 SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS

STARS is a technology development, integration, and transition program to demonstrate a process-driven, domain-specific, reuse-based approach to software engineering (also known as megaprogramming) that is supported by appropriate tools and environment technology.

The goal of the STARS program is to increase software productivity, reliability, and quality by synergistically integrating support for modern software development processes and modern reuse concepts into state-of-the-art Software Engineering Environment (SEE) technology. To meet that goal, STARS has set the following objectives for accomplishment during the 1992–94 time frame.

- Software Reuse—Establish the basis for a paradigm shift to reuse based development
- Processes—Establish capabilities for tailoring process definition and management
- Environments—Establish adaptable, commercially viable SEE solutions that are available on multiple vendors' platforms, are built upon open architecture industry standards, and include automated support for process management and software reuse
- Demonstration/Validation—Demonstrate that the STARS integrated reuse, process, and SEE solutions can be used in actual practice to increase the quality and life-cycle supportability of DOD software products

STARS technical development addresses new areas. Therefore, STARS program management selected three prime contractors—Boeing, IBM, and Unisys—to reduce risk and accelerate acceptance of changing technology. Combining the efforts of three prime contractors will enable cooperative work to be accomplished from a very broad experience base. Furthermore, use of multiple prime contractors and their subcontractors will help accelerate the shift to megaprogramming in other companies.

7.8.1 Reuse

STARS management is working to establish a basis for a paradigm shift to reuse-based development. As part of the activities, technical, management, cultural, and acquisition-related issues are being considered with the goal of reducing the adoption risks in transitioning to reuse-based software engineering. Areas of work include mission domain analysis; asset acquisition; asset classification approaches; asset documentation; software architectures; and asset browsing, storage, recovery, and testing techniques. The planned results of the STARS reuse approach will be

a reuse-based software engineering concept of operation with a modular description of reuse processes associated with various user roles (e.g., domain analyzer, asset certifier, asset cataloger); a reuse library open architecture framework; and asset library mechanisms that support the acquisition, classification, browsing, retrieval, and general management of reusable assets.

7.8.2 Process

STARS will establish capabilities for process definition and management that will show the value of good process concepts, good process definition, process tailoring, and process support in the environment as a vehicle to improve quality, productivity, and reliability. Process definition and tailoring capabilities will support the SEI capability maturity model.

7.8.3 Environment

STARS management is working with framework providers, tool vendors, and standards organizations to ensure that commercially viable environment infrastructures (frameworks) are extensible and robust and conform to open architecture standards. Framework-based environments serve as integration platforms upon which tools, services, and functional capabilities can be integrated to support software development within the context of megaprogramming.

7.8.4 Demonstration

To measure the success of STARS technologies, several demonstration projects will be initiated that will use STARS technologies to develop operational mission-critical applications in Ada. The projects will be designed to provide a pragmatic measure of the progress STARS has achieved in developing and introducing new software engineering approaches and to provide realistic and useful feedback to the technology developers. Demonstration projects are currently being selected; execution of these projects is scheduled to begin in October 1993.

7.9 CORPORATE INFORMATION MANAGEMENT

CIM is the initiative through which DOD will integrate and strengthen central management of the Defense Information Management Program. The goal of the CIM initiative is to improve the effectiveness and efficiency of business processes in DOD by integrating and streamlining functional requirements and by using information technology to implement the improved business operations that result.

The Secretary of Defense assigned to the Office of the Assistant Secretary of Defense (OASD(C3I)) the responsibility for establishing an organization to implement CIM throughout DOD. Pursuant to this direction and in accordance with the "Plan for Implementation of Corporate Information Management in DOD,"

approved by the Deputy Secretary on 14 January 1991, OASD(C3I) established a Directorate of Defense Information (DDI), which is responsible for the following:

- Developing and promulgating information management policies
- Implementing information management processes, programs, and standards
- Integrating the principles of information management into all of DOD's functional activities

This responsibility applies to information technologies and architectures, software, systems development methods and tools, information technology and data standards, and Automatic Data Processing (ADP) equipment acquisition processes. It does not include equipment and software that are an integral part of a weapon or weapons system and related test equipment.

By applying CIM principles, managers of functional activities will be able to streamline business methods and business processes, develop sound business cases and functional economic analyses of their activities and supporting information technology, and provide other improvements in the effectiveness and efficiency of the functional activities. The DDI OASD(C3I) will develop and promulgate guidance on the common models, tools, and methodologies to be used by functional personnel in performing their responsibilities for the management of information related to their functions. CIM also supports the goals of the July 1989 Defense Management Report to the President.

7.10 SOFTWARE ENGINEERING INSTITUTE

SEI is a Federally Funded Research and Development Center (FFRDC) under contract to DOD through Carnegie-Mellon University (CMU). It was established to identify the software engineering needs of the DOD community and provide products and services that enable DOD to make lasting improvements in its ability to acquire, develop, and maintain software-dependent systems; reduce the risk of making such improvements; and educate people to do so more effectively.

SEI's product strategy is to focus on technical areas that have the following characteristics:

- Meet the software engineering needs of a broad segment of DOD
- Can be transitioned through the U.S. infrastructure
- Can be readily adopted by qualified DOD organizations

- Will improve DOD's software engineering capability
- Will yield a demonstrably positive return on investment to the DOD organization
- Will reduce DOD's risk in adopting the underlying technology

To support this strategy, SEI is organized into the Technology Division and the Products and Services Division, each of which is further subdivided into technical areas. The technical areas represent projects that meet the software engineering needs of a broad segment of DOD and support the SEI strategy. The information below briefly describes the projects in each division.

7.10.1 Technology Division

The Technology Division comprises programs and projects in the areas of process, real-time distributed systems, engineering techniques, and risk management. It also has a special products subdivision.

7.10.1.1 Process Program

This program focuses on improving the software development process. Projects conducted under the program are assessing the actual practice of software engineering in the defense community, training organizations to gain management control over their software development processes, supporting the use of quantitative methods and measures as a basic process improvement, and developing improved methods for software process management. Included in this program are the following:

- **Software Capability Maturity Model (CMM)** provides software organizations with guidance on how to gain control of their process for developing and maintaining software and how to evolve toward a culture of software engineering excellence. The CMM was designed to guide software organizations in selecting and aggressively pursuing process improvement strategies by determining current process maturity and identifying the few issues most critical to software quality. Organizations can steadily improve their organization-wide software process to enable continuous and lasting gains in software process capability.
- **Software Capability Evaluation (SCE) Project** is responsible for refining a method for evaluating the software process capability of contractors. The project transitions this method to acquisition organizations that want to reduce program risks and improve their software suppliers' product quality and performance within cost and on schedule. This evaluation can be used as part

of a program's Request for Proposals (RFP) to ensure that the capability of the winning contractor meets the program requirements.

- **Software Acquisition Process Development Project** helps software organizations launch effective process improvement programs, characterizes and reports on the software engineering capabilities of defense contractors, and defines priority needs for software process improvement in the defense community.

- **Software Process Measurement Project** is advocating the use of measurement in the practice of software development and management. Toward that end, it coordinates a steering committee, two working groups, and a best-practices study, all of which are devoted to encouraging organizations to use quantitative methods to improve their software development processes.

- **Software Process Definition Project** supports process improvement through the maturation of the methods and technology associated with software engineering process definition. The project is developing the capabilities required to support the definition and evolution of software processes within an organization.

7.10.1.2 Real-Time Distributed Systems Program

The goal of this program is to improve the development of real-time distributed systems by integrating software engineering with systems engineering and reducing the risks associated with new technology. Projects within this program include the following:

- **Rate Monotonic Analysis for Real-Time Systems Project** is demonstrating how to design and implement real-time systems using analytic scheduling algorithms.

- **Real-Time Embedded Systems Test Bed Project** is collecting, classifying, generating, and disseminating information about software development for real-time embedded systems.

- **Distributed Systems Project** is developing tools and a methodology for building distributed, large-grained, concurrent applications running on heterogeneous machine networks. The project has developed Durra, a language for describing distributed applications as a set of task descriptions and type declarations that prescribes a way to manage the resources of the network.

- **Systems Fault Tolerance Project**, which is in the feasibility stage, was recently initiated to promote the use of fault tolerance in the implementation of dependable or safety-critical systems. Work is underway to characterize the

state of the art in fault tolerance technology, to characterize the state of the practice in applying fault tolerance techniques to actual systems, and to identify barriers to the more widespread use of fault tolerance.

- **Ada 9X Project** is a response to the AJPO's decision that a revision of the Ada language standard is required to maintain it as a standard (ANSI/MIL-STD-1815A). (This revision process is commonly referred to as Ada 9X.) The purpose of this project is to identify and evaluate potential areas for revising the Ada standard based on the experiences of software developers and compiler implementors. This project is providing an organizational framework to help guide revision activities.

- **User Interface Project** is developing Serpent, a user interface management system. Serpent separates the concerns of the user interface from those of the application, which allows integration of I/O technologies without modification to the functional portion of the application.

7.10.1.3 Engineering Techniques Program

The primary objective of the Engineering Techniques Program is to improve the practice of software engineering by improving individual and team productivity through the identification and transition to practice of emerging software technology. Promoting appropriate use of this technology supports SEI's effort to transform software development from an ad hoc, labor-intensive activity to a technology-supported engineering discipline. Projects within this program include the following:

- **Software Architecture Design Principles Project** has as its goal to develop a fundamental understanding of structures for the software architecture level of design. This project is describing basic design elements used in the description, analysis, and development of software systems.

- **Software Architectures Engineering (SAE) Project** provides DOD program offices with improvements to the practice of software engineering by helping develop and insert new architectural and practice models where old models have proved inadequate. To accomplish this goal, the project helps develop a new set of engineering optimizations (goals) in the application area. It also helps set the requirements for models tunable to the new practitioners in the area by extending the architectures to address possibilities precluded by the current model set. Project members are refining and maturing the new model sets by transitioning them to others and providing additional sources of reflection on their use.

- **Software Development Environments Project** is assessing state-of-the-art commercial environment support for configuration management with the goal of moving toward a common configuration management model.

- **Software Process Modeling Project** is investigating techniques for modeling software development and maintenance processes.

- **Domain Analysis Project** is developing and testing methods for performing domain analysis to support software reuse. The primary objectives of the project are to define a process and set of products to support the systematic discovery and exploitation of commonality across related software systems and to apply the process to a domain within the Army Tactical Command and Control System (ATCCS).

- **Domain-Specific Software Architecture Project** is providing solutions to the design problems that characte.ize an application domain. Currently, project members are focusing on the training simulators systems domain in an effort to increase the use of modeling in the development of system architectures.

- **Requirements Elicitation Project** has as its goal to demonstrate, develop, and promote advanced methods and technologies that are applied to eliciting requirements and prototyping.

- **Requirements Engineering Project** has been revising the project plan based on the comments from informal reviews within SEI and from the presentation made at a U.S. Army Communications Electronics Command (CECOM) meeting on June 12. The project is planning to have a formal review in July.

- **Advanced Video Techniques for Imaging Project** is using advanced video techniques for imaging.

- **Computer-Aided Software Engineering (CASE) Technology Project** focuses on improving the ability of SEI sponsors and affiliates to make informed decisions about adopting CASE technology and to improve their practice in using it. The project members also will provide information to tool vendors on use of current tools and need for improvement.

7.10.1.4 Risk Management Program

The Risk Management Program is focusing on improving the management of risk in DOD programs involving software-dependent systems. In this context, risk management is the identification, confrontation, and resolution of software-related risks. The program objectives are to establish a foundation for addressing

software-related risk and to strengthen the ability of organizations in the software community to evaluate and manage software-related risk.

7.10.1.5 Special Projects

SEI also is involved in several special projects, including the following:

- **Empirical Models Function** supports transition management of SEI technology projects by providing market research methods and materials, conducting surveys, and evaluating events or validating products of SEI projects.

- **Transition Methods Project** will evolve a set of methods for planning, implementing, and assessing transition activities that will be useful for technology producers and consumers both inside and outside SEI. Project staff also will provide SEI staff with education and training on technology transition concepts and approaches.

7.10.2 Products and Services Division

The Products and Services Division programs and projects focus on education, security issues, the reliability and adaptability of mission-critical software, and Ada binding to SQL.

7.10.2.1 Master of Software Engineering Degree Program

This program is a joint effort between SEI and the CMU School of Computer Sciences to establish a 2-year professional degree program. The objective of the program, which uses an education paradigm based on a master-apprentice relationship, is to produce a few highly skilled experts in software system development. Its main focus is on software products and the application of principles from computer science and other related disciplines to engineer superior products.

7.10.2.2 Education Program

This program has as its primary objective to increase the number of highly qualified software engineers by rapidly improving software engineering education throughout academic, Government, and industry education communities. To accomplish this objective, Education Program projects focus on accelerating the development of software engineering programs in academia and on enhancing opportunities for the continuing education of practitioners. Projects within this program include the following:

- **Software Engineering Curriculum** is developing model curricula, promoting the growth of graduate software engineering programs in the academic community, investigating the feasibility of undergraduate programs, and working to increase the amount of software engineering content in both undergraduate and

graduate computer science programs. The project produces education materials, including the Academic Series (formerly part of the Video Dissemination Project), which is a set of videotaped graduate-level courses on software engineering.

- **Continuing Education** interacts with industry and Government to increase the availability of high-quality educational opportunities in software engineering topics for software practitioners and executives. The project produces the Continuing Education Series and the Technology Series. The Continuing Education Series consists of video-based courses designed for clients' in-house education and executive offerings designed for decision makers involved in improvement efforts. The Technology Series provides stand-alone presentations that promote awareness of emerging issues and leading-edge technologies.

7.10.2.3 Software Engineering Institute Services

SEI Services houses the Computer Emergency Response Team Coordination Center (CERT/CC). This center supplements existing mechanisms by which informally organized experts deal with and prevent computer emergencies. The CERT/CC at SEI supports two different communities: (1) Internet users and (2) developers of technology that is available on the network, such as UNIX and networking software. The CERT/CC provides a dependable 24-hour point of contact for security issues and allows rapid communication during emergencies. It also raises constituents' awareness of security issues and helps individual organizations improve the security of their systems. The CERT/CC also maintains a highly secure repository of information for team members and cultivates close ties with researchers in the area of trusted systems to improve the security of existing systems.

7.10.2.4 Defense Advanced Research Projects/Software Technology for Adaptable, Reliable Systems Program

As software becomes a more critical component of defense systems, the demand for reliability and adaptability in mission-critical software is growing more rapidly than the defense community's ability to produce it. To address this problem, DARPA launched the STARS Program. The program goal is to increase productivity and the reliability of defense software while also achieving reductions in the cost of software production and maintenance. To accomplish this goal, STARS will integrate major thrusts in reuse, software process, and software engineering environments and incorporate a commercialization strategy. Projects within this program include the following:

- **Binding of Ada and SQL Project**, initiated at the request of AJPO, has investigated the problem of binding the Ada programming language with the

SQL database language. The solution to this problem was the specification of the SQL.

- **Issues in Ada Adoption Project** conducts activities in concert with DOD components to demonstrate and encourage the use of Ada and Ada-related technology.

- **STARS Support Project** is helping implement and promote technology developed as part of the STARS Program. The project is working with the STARS Program Office and the three prime contractors to provide system engineering and technical assistance.

7.11 PLANS

Currently, several DOD planning initiatives relating to Ada and/or Ada-related technologies are in process. The subsections below provide synopses of these activities.

7.11.1 Software Action Plan

The 4 June 1991 Director of Defense Research and Engineering (DDR&E) memorandum that established the SWAP enunciated the purpose of the SWAP. This memorandum charges the SWAP Working Group to develop and implement "an integrated technology and management plan to ensure more cost-effective support of weapons systems and related test equipment systems within DDR&E's purview."

The primary focus of the SWAP is DOD weapons systems software (i.e., the software that fights the battles). This software is integral to ground, sea, air, and space superiority. DDR&E and the DOD components have a major responsibility to acquire this software correctly and ensure that it operates and is supported properly.

The 12 August 1991 Deputy Secretary of Defense memorandum, "Strengthening Technology and Acquisition Functions," indicates that DDR&E is responsible for all technology activities supported by 6.1, 6.2, and 6.3A funds, which means that the technology addressed in the SWAP applies to all of DOD. A separate DOD SWTP has been drafted and is being refined to address the technology aspects of the SWAP.

Other software-intensive functions within DDR&E's purview are engineering and scientific software that supports weapons systems and technology activities and test-equipment systems. Also, to the extent that the SWAP can pursue common collaborative software actions with other DOD organizations, particularly OASD(C3I), it will operate in a much wider scope.

In concert with complementary software management initiatives elsewhere in DOD, the SWAP is designed to meet the following objectives by the year 2000:

- Reduce equivalent software system life-cycle costs by a factor of 2
- Reduce software problem rates by a factor of 10
- Significantly expand DOD capabilities via software

The reduction in software problem rates refers both to error rates for delivered DOD software and to the level of incidence of software problems as the major cause of DOD system acquisition problems.

Capabilities resulting from the expansion efforts will enable DOD to reap the full benefits of information processing and software technology, both as a force multiplier and as a source of improved corporate management.

7.11.2 Software Technology Plan

The DOD SWTP is designed to define and justify a coordinated set of DOD software science and technology actions and investments that will meet DOD needs for improved software functionality and bring future DOD software costs under control. To ensure that DOD long-range technology needs and opportunities are addressed, the scope of the SWTP extends beyond the year 2000. The plan covers the 15-year period of DOD software technology investments between fiscal years 1992 and 2007, with greater detail provided for the first 5 years.

This plan, developed under the auspices of the DDR&E as part of the DDR&E SWAP, will be reassessed and updated biannually by a steering group representing major DOD stakeholders. The plan will be implemented by the military departments and agencies cited therein for each of the specific software technology efforts.

7.11.3 Computer Resources Strategy

The Computer Resources Strategy provides broad policy guidance to the DON for future direction in the transition to modern acquisition, systems engineering, and business management practices for acquiring and supporting its computing resources. The strategy encourages the DON community to pursue vigorously continuous improvements in the quality of new and existing systems.

GLOSSARY

ABET	Ada-Based Enhancements for Test
ACEC	Ada Compiler Evaluation Capability
ACM	Association for Computing Machinery
ACVC	Ada Compiler Validation Capability
AdaIC	Ada Information Clearinghouse
AdaJUG	Ada Joint (Services) Users Group
Ada PSE	Ada Programming Support Environment
ADP	Automatic Data Processing
AES	Ada Evaluation System
AFATDS	Advanced Field Artillery Tactical Data System
AFB	Air Force Base
AFSC	Air Force Systems Command
AI	Artificial Intelligence
AIE	Ada Integrated Environment
AIS	Automated Information System
AIU	Acoustic Interface Unit
AJPO	Ada Joint Program Office
ALS	Ada Language System
ALS/N	Ada Language System/Navy
AMMWS	Advanced Millimeter Wave Seeker
AMPS	Advanced Message Processing System
ANSI	American National Standards Institute
AP	Acquisition Plan
APP	Application Portability Profile
ASEET	Ada Software Engineering Education and Training
ASIS	Ada Semantic Interface Specification
ASP	Acquisition Strategy Plan
ASR	Ada Software Repository
AST	Advanced Systems Technology
ASW	Anti-Submarine Warfare
ASWSOW	Anti-Submarine Standoff Weapon
AT&T	American Telephone & Telegraph
ATCCS	Army Tactical Command and Control System
ATF	Advanced Tactical Fighter
ATIP	Ada Technology Insertion Program
ATRIM	Aviation Training and Readiness System
AVF	Ada Validation Facilities

BAFO	Best and Final Offer
BBS	Bulletin Board System
BP	Backplane
C3I	Command, Control, Communications, and Intelligence
C4I	Command, Control, Communications, Computers, and Intelligence
CAB	Common Ada Baseline
CAIS	Common Ada PSE Interface Set
CALS	Computer-Aided Logistics Support
CAMP	Common Ada Missile Packages
CASE	Computer-Aided Software Engineering
CAS REPS	Casualty Reporting System
CAUWG	Commercial Ada Users Working Group
CAXI	Common Ada XWindow Interface
CC&I	Command, Control, and Intelligence
CCITT	International Consultative Committee for Telegraph and Telephone
CCP	Code Counting Program
CCS	Combat Control System
CDA	Central Design Agency
CDIF	CASE Data Interchange Format
CDPA	Central Design Programming Activity
CDR	Critical Design Review
CDRL	Contract Data Requirements List
CECOM	Communications Electronics Command
CERT/CC	Computer Emergency Response Team Coordination Center
CFE	Contractor-Furnished Equipment
CGI	Computer Graphics Interface
CI	Configuration Item
CIF	Central Issue Facility
CIM	Corporate Information Management
CLNP	Connectionless Network Protocol
CLOC	Compiled/Assembled Lines of Code
CMM	Capability Maturity Model
CMS-2	Compiler Monitor System-2
CMU	Carnegie-Mellon University
CMU/SEI	Carnegie-Mellon University/Software Engineering Institute
COBOL	Common Business Oriented Language
COE	Common Operating Environment

COMNAVCOMTELCOM	Commander, Naval Computer and Telecommunications Command
COMSPAWARSYSCOM	Commander, Space and Naval Warfare Systems Command
COTS	Commercial Off-The-Shelf
CPDL	Computer Program Development Laboratory
CPU	Central Processing Unit
CREASE	Catalog of Resources for Education
CRISD	Computer Resource Integrated Software Document
CRLCMP	Computer Resources Life-Cycle Management Plan
CRSS	C3I Reusable Software System
CSC	Computer Sciences Corporation
CSCI	Computer Software Configuration Item
CSS	Centralized Structure Store
CSS	Computer Sciences School
CSU	Computer Software Unit
CWG	Coordinator Working Group
D&V	Demonstration & Validation
DAB	Defense Acquisition Board
DACS	Data and Analysis Center for Software
DARPA	Defense Advanced Research Projects Agency
DBMS	Database Management System
DC	Device Coordinate
DCDS	Distributed Computing Design System
DDI	Directorate of Defense Information
DDN	Defense Data Network
DDR&E	Director of Defense Research and Engineering
DE1	Data Elements in the Source
DEMVAL	Demonstration Evaluation
DFCS	Digital Flight Control System
DID	Data Item Description
DISA	Defense Information Systems Agency
DOD	Department of Defense
DON	Department of the Navy
DP/DGU	Distributed Processor/Display Generator Unit
DSRS	Defense Software Repository System
DTC II	Desk Top Computer II
DTIC	Defense Technical Information Center
DWS	Defense Weapons System

ECLD	Embedded Comment Lines in Data
ECLS	Embedded Comment Lines in Source
ECM	Electronic Countermeasures
ECMA	European Computer Manufacturing Association
EDL	Event-Driven Language
EMR	Extended Memory Reach
ENB	Engineering Notebook
EPROM	Erasable Programmable Read Only Memory
EP	Enhanced Processor
ERA	Entity Relationship Attribute
ESD	Electronic Systems Division
ESM	Electronic Support Measure
FAR	Federal Acquisition Regulations
FAU	Fin Actuator Unit
FCDSSA	Fleet Combat Direction System Support Activity
FFP	Firm Fixed Price
FFRDC	Federally Funded Research and Development Center
FIPS	Federal Information Processing Standards
FRAWG	Front Range Ada Working Group
FSD	Full-Scale Development
FTAM	File Transfer, Access, and Management
ftp	File Transfer Protocol
43RSS	AN/UYK-43 Run-Time Support System
GAO	General Accounting Office
GEU	Guidance Electronics Unit
GFE	Government-Furnished Equipment
GFS	Government-Furnished Software
GKS	Graphical Kernel System
GNCP	Guidance, Navigation, and Control Program
GOSIP	Government Open Systems Interconnection Profile
GPEF	Generic Package of Elementary Functions
GPPF	Generic Package of Primitive Functions
GPO	Government Printing Office
GRACE™	Generic Reusable Ada Components for Engineering
GSIS	Graphics System Interface Standard
GTRIMS	Ground Controller Training System
HOL	High Order Language
HPBP	High-Performance Backplane
HPP	High-Performance Processor

IBM	International Business Machines
ICE	In-Circuit Emulator
IEC	International Electro-Technical (Committee)
IEEE	Institute of Electrical and Electronics Engineers
IGES	Initial Graphics Exchange Specification
ILSP	Integrated Logistic Support Plan
IMU	Inertial Measurement Unit
INEL	Idaho National Engineering Laboratory
InProc	In Processing
I/O	Input/Output
IPR	In-Process Review
IPS	Integrated Project Summary
IPSE	Integrated Project Support Environment
IRAC	International Requirements and Design Criteria
IRDS	Information Resource Dictionary System
IRS	Interface Requirements Specification
ISA	Instruction Set Architecture
ISDN	Integrated Services Digital Network
ISEA	In-Service Engineering Activity
ISEE	Integrated Software Engineering Environment
ISO	International Standards Organization
ISSC	Information System Software Center
ITS	Integrated Test Software
IV&V	Independent Verification and Validation
JCS	Joint Chiefs of Staff
JTC	Joint Technical Committee
KAPSE	Kernel Ada Programming Support Environment
LAN	Local Area Network
LCM	Life-Cycle Maintenance
LCSA	Life-Cycle Support Activity
MAPSE	Minimal Ada Programming Support Environment
MCCDC	Marine Corps Combat Development Command
MCCR	Mission-Critical Computer Resources
MCCRES	Marine Corps Combat Readiness Evaluation System
MENS	Mission Element Need Statement
MEPS	Message Edit Processing System
MHS	Message Handling Service
MIL-HDBK	Military Handbook

MIL-STD	Military Standard
MIMMS	Marine Corps Integrated Maintenance Management System
MIS	Management Information System
MMS	Minimum Mode Software
MOA	Memorandum of Agreement
MOTS	Military Off-The-Shelf
NAC	Naval Avionics Center
NADC	Naval Air Development Center
NARDAC	Navy Regional Data Automation Center
NASA	National Aeronautics and Space Administration
NASEE	NAVAIR Software Engineering Environment
NATO	North Atlantic Treaty Organization
NAUG	Navy Ada Users Group
NAVAIR	Naval Air Systems Command
NAVDAC	Navy Data Automation Command
NAVSEA	Naval Sea Systems Command
NAVSWC	Naval Surface Warfare Center
NCS	Network Computing Service
NCTAMS	Naval Computer and Telecommunications Area Master Station
NCTAMS LANT	NCTAMS Atlantic
NCTAMS EASTPAC	NCTAMS Eastern Pacific
NCTC	Naval Computer and Telecommunications Command
NCTS	Naval Computer and Telecommunications Station
NDC	Normalized Device Coordinate
NDI	Nondevelopmental Item
NGCR	Next Generation Computer Resources
NISBS	NATO Interoperable Submarine Broadcast System
NIST	National Institute of Standards and Technology
NISMC	Naval Information System Management Center
NOSC	Naval Ocean Systems Center
NSWC	Naval Surface Weapons Center
NTIS	National Technical Information Service
NUSC	Naval Undersea Command
NWRC	Navy Wide Reuse Center
NWSUS	Navy WWMCCS Site-Unique Software
OAS	Offensive Avionics System
OASD	Office of the Assistant Secretary of Defense
OCD	Operational Concept Document

OFPS	Operational Flight Program Size
OMU	Operational Mock-up
OOD	Object-Oriented Design
OORA	Object-Oriented Requirements Analysis
OPE	Open Systems Environment
OPNAVINST	Naval Operations Instruction
OPR	Office of Primary Responsibility
ORG	Organization Chain of Command
OS	Operating System
OSA	Open Systems Architecture
OSE	Open Systems Environment
OSF	Open Software Foundation
OSI	Open Systems Interconnection
OSS	Operations Support System
OSSWG	Operating Systems Standards Working Group
PC	Personal Computer
PCIS	Portable Common Interface Set
PCTE	Portable Common Tool Environment
PDL	Program Design Language
PDR	Preliminary Design Review
PDS	Post-Deployment Support
PDSS	Post-Deployment Software Support
PDU	Pulse Driver Unit
PHIGS	Programmers Hierarchical Interactive Graphics System
PIWG	Performance Issues Working Group
PMC	Project Management Charter
POSIX	Portable Operating System Interface for UNIX
PPBS	Planning, Programming, and Budgeting System
PRR	Product Readiness Review
PSE	Project (or Programming) Support Environment
PSESWG	Project Support Environment Standard Working Group
R&D	Research and Development
RACS	Registration and Access Control System
RADC	Requirements and Design Criteria
RAM	Random Access Memory
RAPID	Reusable Ada Products for Information Systems Development
RCL	RAPID Center Library
RDA	Remote Database Access

RDBMS	Relational Database Management System
RDT&E	Research, Development, Test, and Evaluation
RES	Resources
RFP	Request for Proposals
RLF	Reuse Library Framework
RMC	Reconfigurable Mission Computer
ROM	Read Only Memory
RPC	Remote Process Communication
RSC	Reusable Ada Software Component
RTE	Run-Time Environment
SAE	Software Architectures Engineering
SAFENET	Survivable Adaptable Fiber-optic Embedded Network
SAI	Software Action Item
SAIL	System Avionics Integration Laboratory
SAME	SQL Ada Module Extension
SAMeDL	SQL Ada Module Description Language
SASSY	Supported Activities Supply System
SCCS	Submarine Combat Control System
SCE	Software Capability Evaluation
SCL	Stand-alone Comment Lines
SCMP	System Configuration Management Plan
SCS	Submarine Combat System
SDC-W	Software Development Center, Washington
SDD	System Design Definition
SDE	Software Development Environment
SDF	Software Development Folder
SDIO	Strategic Defense Initiative Organization
SDL	Software Development Laboratory
SDP	Software Development Plan
SDP	System Division Paper
SDR	System Design Review
SDSR	Software Development Status Report
SECNAVINST	Secretary of the Navy Instruction
SECR	Standard Embedded Computer Resources
SEE	Software Engineering Environment
SEI	Software Engineering Institute
SEMP	System Engineering Management Plan
SEO	Software Executive Official
SEPG	Software Engineering Process Group
SGS/AC	Shipboard Gridlock System with Auto-Correlation
SGML	Standard Generalized Markup Language

SIGAda	Special Interest Group on Ada
SIGSOFT	Special Interest Group on Software Engineering
SIL	System Integration Laboratory
SIP	System Integration Plan
SLOC	Source Lines of Code
SLOCWC	Source Lines of Code Without Comments
SMB	Submarine Message Buffer
SMM	Software Management Metrics
SMP	Software Master Plan
SOW	Statement of Work
SPA	Software Process Assessment
SPAWAR	Space and Naval Warfare Systems Command
SPC	Software Productivity Consortium
SPR	Software Problem Report
SQAP	Software Quality Assurance Plan
SQL	Structured Query Language
SRC	Software Requirements Change
SRR	System Requirements Review
SRS	Software Requirements Specification
SSA	Software Support Activity
STANFINS	Standard Financial System
STARFIARS	Standard Army Financial Accounting and Reporting System
STARS	Software Technology for Adaptable, Reliable Systems
STSC	Software Technology Support Center
SUP	Support Planning
SWAP	Software Action Plan
SWG	Special Working Group
SWTP	Software Technology Plan
TACAMO	Take Charge and Move Out
TACFIRE	Tactical Fire Direction
TADSTAND	Tactical Digital Standard
TC	Target Capacity
TCL	Total Comment Lines
TCP/IP	Transmission Control Protocol/Internet Protocol
TDA	Technical Directive Authority
TDT	Theater Display Terminal
TEMP	Test and Evaluation Master Plan
TEP	Test and Evaluation Plan
TFA	Transparent File Access
TLCSC/LLCSC	Top Level/Lower Level Computer Software Component

TLOC	Total Lines of Code
TOES	Telephone Order-Entry System
TSGCEE	Tri-Service Group on Communications and Electronics Equipment
UIMS	User Interface Management System
ULLS	Unit Level Logistics System
USMC	U.S. Marine Corps
USW	Undersea Warfare
VADS	Verdix Ada Development System
VDI	Virtual Device Interface
VHSIC	Very High-Speed Integrated Circuit
VRC	Virtual Reference Coordinate
VSR	Validation Summary Reports
VT	Virtual Terminal
WAdaS	Washington Ada Symposium
WAM	WWMCCS ADP Modernization
WBS	Work Breakdown Structure
WC	World Coordinate
WIS	WWMCCS Information System
WWMCCS	World Wide Military Command and Control System

References

Ada Joint Program Office, "Common Ada Programming Support Environment (Ada PSE) Interface Set (CAIS) (Revision A)," 6 April 1989.

Archer, T. S., "Managing the Ada Conversion and Integration of Mission Critical Defense Systems," 9th Annual National Conference in Ada Technologies, March 1991.

Assistant Secretary of Defense (Command, Control, Communications, and Intelligence [C3I], Memorandum: "Plan for Implementation of Corporate Information Management in DoD," 14 January 1991.

Association for Computing Machinery (ACM) Special Interest Group on Ada (SIGAda), *Implementing the DOD-STD-2167 and DOD-STD-2167A Software Organizational Structure in Ada*, August 1990.

Boehm, B. W., "Ada COCOMO and the Ada Process Model," *Proceedings, Fifth COCOMO Users Group Meeting*, SEI/CMU, October 1989.

Boehm, B.W., *Software Engineering Economics*, Prentice Hall, Englewood Cliffs, NJ, 1981.

Booch, G., *Software Components with Ada*, Benjamin/Cummings, Menlo Park, CA., 1987.

Booch, G., *Software Engineering with Ada*, Benjamin/Cummings, Menlo Park, CA., 1983.

Carnegie-Mellon University/Software Engineering Institute, *Software Capability Evaluation Overview*, 19-21 March 1991.

Carnegie-Mellon University/Software Engineering Institute, *Software Process Assessment*, September 1990.

Carnegie-Mellon University/Software Engineering Institute, *Software Metrics* (SEI-CM-12-1.1), December 1988.

Cohen, S. G., *Ada 9X Issues—Reuse Study*, SEI (Draft), 1989.

Conn, R., "Overview of the DOD Ada Software Repository," *DR. DOBB'S JOURNAL*, February 1986.

Deputy Secretary of Defense, Memorandum: "Strengthening Technology and Acquisition Functions," 12 August 1991.

Dyson, P. B., *Metrics Application Plan for the Take Charge and Move Out (TACAMO) Message Processing System: Technical Report* (prepared for the Naval Air Development Center, Warminster, PA, under Contract Number: N62269-86-C-0415), 27 April 1989.

EVB, "Creating Reusable Ada Software," *Proceedings of the Conference on Software Reusability and Maintainability*, The National Institute for Software Quality and Productivity, Inc., Tysons Corner, VA, March 1987.

Feiler, P. and G. Downey, *Tool Version Management Technology: A Case Study* (CMU/SEI-90-TR-26, Ada 235639), 1990.

FIPS-PUBS 127-1, *Database Language SQL*, 2 February 1990 (incorporates ANSI X.3 135-1989, *Database Language—SQL With Integrity Enhancement* and ANSI X.3 168-1989, *Database Language—Embedded SQL*).

Hitchon et al., *Introduction to CAIS (MIL-STD-1838A)*, 1989.

Humphrey, W. S., *Managing the Software Process*, Addison-Wesley, Reading, MA, 1989.

Humphrey, W. S., *Characterizing the Software Process: A Maturity Framework*, (CMU/SEI-87-TR-11, ESD-TR-87-112), June 1987.

Humphrey, W. S. and W.L. Sweet, *A Method for Assessing the Software Engineering Capability of Contractors* (CMU/SEI-87-TR-23, ESD-TR-87-186), September 1987.

ITT Research Institute, *Available Ada Bindings*, January 1992.

Lesslie, P. A., R. O. Chester, and M. F. Theofanos, *Guidelines Document for Ada Reuse and Metrics* (DRAFT), Martin Marietta Energy Systems, Inc., Oak Ridge, TN, 1988.

McDonnell Douglas Missile System Company, *Common Ada Missile Packages—Leading the Way in Software Reuse* (videotapes), available from McDonnell Douglas, St. Louis, MO.

McNicholl, D. G. et al., *Common Missile Packages (CAMP)*, Vol. I: *Overview and Commonality Study Results* (AFATL-TR-85-93), McDonnell Douglas, St. Louis, MO, 1986.

MIL-HDBK 287, "A tailoring guide for DOD-STD-2167A," 29 February 1988.

MIL-STD-1838A, "Common Ada Programming Support Environment (APSE) Interface Set," 30 September 1989.

Musgrove, R. G., *Workshop on Commonality in Computing for NASA Flight Systems*, Lyndon B. Johnson Space Center, Houston, TX, 1987.

National Aeronautics and Space Administration, *Software Engineering Laboratory (SEL) Guidebook.*

Prieto-Diaz, R. and P. Freeman, "Classifying Software for Reusability," IEEE Computer Society Press, Vol. 4, No. 1, Los Alamitos, CA, January 1987.

San Antonio I, Panel I, *Software Metrics Implementation Final Report*, 11 December 1991.

San Antonio I, Panel VII, *JLC Software Workshop Final Report*, 1 February 1991.

Shlaer, S. and S. Miller, "An Object-Oriented Approach to Domain Analysis," *Software Engineering Notes*, Vol. 14, No. 5, November 1989.

Software Productivity Consortium, *Ada Quality and Style, Guidelines for Professional Programmers*, Van Nostrand, Reinhold, NY, 1989.

U.S. Air Force, *Air Force Systems Command Software Quality Indicators: Management Quality Oversight*, AFSCP 800-14, January 1987.

U.S. Air Force, *Air Force Systems Command Software Management Indicators: Management Insight*, AFSCP 800-43, January 1986.

U.S. Department of Commerce, National Institute of Standards and Technology, "Application Portability Profile (APP) The U.S. Government's Open System Environment Profile OSE/1 Version 1.0," NIST USGPO, Washington, D.C., 1991

U.S. Department of the Navy, *Interim Department of the Navy Policy on Ada*, 24 June 1991.

Weiderman, N., *Ada Adoption Handbook, Compiler Evaluation and Selection*, Version 1.0 (CMU/SEI 89-TR-13, ESD-TR-89-12), 1989.